Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Master's thesis

Data Efficient Language Model Pretraining

in the Language Technology (LT) group

by

Elisabeth Fittschen
born on 20.01.1999
Matriculation number: 7657939
Field of study: Computer Science

submitted August 17, 2026

Supervisor: Tim Fischer
First Reviewer: Prof. Dr. Chris Biemann
Second Reviewer: Dr. Seid Yimam

Contents

1 Introduction

1.1 Motivation

Modern large language model (LLM) pretraining is structured around the discovery that performance continues to predictably increase when scaling model parameters and training dataset tokens (Kaplan et al., 2020). The effects of these "scaling laws" can be seen in Figure 1.1, as time passes models are being trained on more an more data. Even more modern models like Llama 3 (Grattafiori et al., 2024) and DeepSeek (DeepSeek-AI, Guo, et al., 2025) were trained on upwards of 14 trillion tokens.

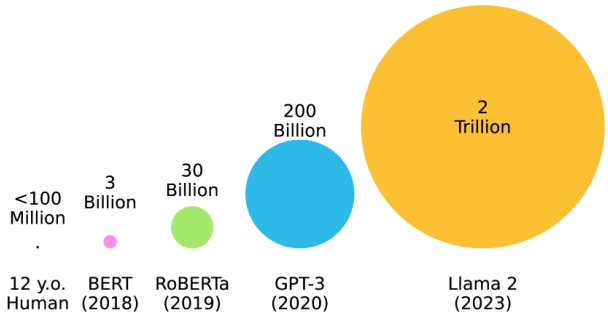

Figure 1.1: A visualisation of model training data scales.

This thesis explores the 'data-efficient' training regime, which aims to train performant language models in data-constrained settings. There are three primary considerations that motivate this direction:

Plausibility Humans require about 5 orders of magnitude less data, as seen in Figure 1.1. It stands to reason that it should theoretically be possible to train models with basic language competence using similar scales of data.

pplications There are many downstream applications which would benefit from paradigms for pre-training over controlled datasets. For instance, when doing scientific research there is a big confound, which raises the question- does the model's knowledge of all major theories (related to the object of study) biases the responses it generates? Similarly when generating a customer service-facing model, or a model to interact with children in educational settings, there is strong incentive to invest in methods that allow us to tightly control the behaviors and knowledge the model is theoretically able to reproduce.

Outlook Computational resources have historically grown much faster than training data. It is estimated that compute spent on pre-training grows by a factor of $4\times$ year, in contrast to web text, which increases only by 1 $03\times$. If this trend continues, data may eventually become the critical bottleneck, making data efficiency an increasingly relevant research direction.

The Master's thesis will approach this problem using the BabyLM Challenge as a test bed. It will adopt the BabyLM training set (Section 2.3) and its zero shot evaluation suite (Section 2.3).

1.2 pproaches

1.2.1 Epoch Scaling

Prior research from the 2024 edition of the BabyLM Shared Task (Tastet and Timiryasov, 2024; L Charpentier and Samuel, 2024) shows that tuning weight decay is very important in the multi-epoch setting, and optimal values are much higher than what is typically observed otherwise. To characterise this further, we ran a number of scaling experiments, hoping to extend prior work published on this topic of scaling laws (Muennighoff et al., 2025). Concurrently Kim et al. (2026) investigates a closely related scaling phenomenon. Our results are consistent with their conclusions. We verify the discovery of Wang and Aitchison, 2025 that the optimal hyper-parameter are consistent in $lr \times wd$. We show that in non mu-p initialized models, consistency with this rule is related to normalized parameter behavior. We explore the research question of why randomly shuffled epochs have a cyclic dev loss movement.

1.2.2 Loss ugmentation

The best-performing models of the past years' BabyLM challenge have typically used some variation of masked language modeling (MLM) as a loss signal. One hypothesis for why MLM outperforms other approaches has been that it provides a more diverse loss signal in comparison to causal language modeling (CLM). While CLM can only apply loss to a token given a fixed context, MLM models can generate many more masked views per datapoint. MLM models, however, need to perform a forward pass over the whole sequence to (per convention) only predict 15% of sentence tokens. This likely makes MLM models particularly inefficient in terms of forward/backward passes, which is the metric used to restrict compute in the 2025 edition of the BabyLM challenge. Motivated by this, we propose using the permutation language modeling (PLM) objective, first introduced in XLNet (Z Yang et al., 2019), which leverages bidirectional context to extract a richer training signal from these datapoints. XLNET traditionally only trains on the last 16% of tokens, due to the noise present when predicting earlier tokens. We will explore smaller sequence pertubation orders, and implement a Permutation Model using a modern Llama transformer.

2 Background

2.1 BabyLM Shared Task

The BabyLM shared task, running since 2023 (Warstadt, Mueller, et al., 2023), focuses on training language models on human plausible amounts of data. This task involves the use of training datasets consisting of spoken language interactions, specifically transcribed conversations between children and parents (refer to Table 2.2 for further details). The shared task has a number of tracks:

- **Strict**: 100M tokens of text training data
- **Strict Small**: 10M tokens of text training data
- **Interactive**: 100M tokens or less, which may be interactively generated
- **Multimodal**: 100M tokens text buget, paired with images

In this work, we primarily focus on standard language model training, which falls in the realm of the strict and strict small task. For most of the previous iterations of the BabyLM shared task, there were no restrictions on how long the language models could be trained for. However, starting from 2025, the shared task introduced a limit where the model was allowed to only train on forward passes spanning $10\times$ of the data limit, which translates to effectively 10 epochs. It is possible that this restriction was put in place to promote more equal access to the competition, as the 2024 winners (LGG Charpentier and Samuel, 2023; L Charpentier and Samuel, 2024) trained their model using 128 AMD MI250X GPUs in parallel.

This thesis will use the BabyLM dataset for training, and a chosen number of the evaluation methods typically used to assess the models in this shared task setting. Further details are provided in Section 2.2 and Section 2.3 respectively.

2.2 Datasets

This work uses the 100M word dataset provided in the BabyLM challenge. Unlike in the BabyLM challenge, we use tokens rather than words as the base unit of measurement. While BabyLM restricts words (so as to avoid favoring uncommon tokenizer setups), this is not a fundamental issue in our setup because we do not vary the tokenizer and use the standard `ltg/gpt-bert-babylm-base`[1] tokenizer throughout. This tokenizer has a vocabulary size of 16,384 tokens and was trained on a corpus of comparable size. It was chosen because it works well with the official BabyLM tool suite. An overview of the corpus components is shown in Table 2.1. Note that a 100M words corresponds to approximately 176M tokens– by extension, training over a 100M tokens would then be equivalent to training over roughly 56.8M words.

Dataset	Domain	# Docs	Tokens (M)	Proportion
CHILDES	Child-directed speech	9,888	69.53	39.4%
British National Corpus (BNC), spoken portion	Spoken dialogue	699	10.79	6.1%
Standardized Project Gutenberg Corpus	Written English	563	38.70	21.9%
OpenSubtitles	Movie subtitles	3,835	32.73	18.6%
Simple Wikipedia	Wikipedia (Simple English)	116,027	22.37	12.7%
Switchboard Dialog Act Corpus	Dialogue	809	2.25	1.3%
Total	–	131,821	176.36	100%

Table 2.1: Dataset makeup of the tokenized corpus used in this work. Token counts are shown in millions (M).

Corpus content

We will provide a small description and excerpt for all of the contributing datasets.

British National Corpus (BNC) The BNC (BNC Consortium, 2007) is a large, balanced resource of late-20th-century British English containing both written and spoken material. We use the spoken portion for our experiments, which includes spontaneous conversational speech captured as orthographic transcripts. As illustrated in Figure 2.1, these samples are turn-based and contain typical properties of spoken interaction.

BNC excerpt

Well it's just that, you know, a pound, or a hundred pounds today, is not the same as a hundred pounds in a year's time, or two, two years' time.
Right.
So that would be opportunity cost?
That's right, yes, that's right, so, in actual fact, this area here, right, is going to be the discounted sum of all future rural incomes.

Figure 2.1: Representative excerpt from the spoken portion of the BNC dataset.

1. https://huggingface.co/ltg/gpt-bert-babylm-base

Child Language Data Exchange System (CHILDES) CHILDES (MacWhinney, 2000) is a large repository of child language acquisition data, primarily consisting of transcripts of interactions between children and caregivers and/or researchers. These transcripts are commonly distributed in the CHAT format, which encodes speaker tiers (e.g., `*CHI`, `*MOT`, `*INV`) and optional contextual annotations (e.g., activity descriptions), matching the structure shown in Figure 2.2.

CHILDES excerpt

[playing with toys]
***CHI:** fix.
***MOT:** yes it's gotta be fixed.
***MOT:** we'll have a new skirting board on.
***CHI:** fix.
***INV:** is the man gonna come and fix it?
***CHI:** yeah.
***INV:** is he?
***INV:** next week.
***MOT:** hope so.
***CHI:** book.
***INV:** oh which book do you want?

Figure 2.2: Representative excerpt from the CHILDES dataset.

Gutenberg Corpus The Gutenberg Corpus (Gerlach and Font-Clos, 2018) consists of long-form written texts (predominantly books), yielding well-edited narrative prose with complex syntax and extended descriptive passages, exemplified by Figure 2.3.

Gutenberg excerpt

CHAPTER ONE - *TO S VE COMR DE.*
A sharp volley, which ran echoing along the ravine, then another, just as the faint bluish smoke from some hundred or two muskets floated up into the bright sunshine from amidst the scattered chestnuts and cork-trees that filled the lower part of the beautiful gorge, where, now hidden, now flashing out and scattering the rays of the sun, a torrent roared and foamed along its rocky course onward towards its junction with the great Spanish river whose destination was the sea.

Figure 2.3: Representative excerpt from the Gutenberg dataset.

OpenSubtitles The OpenSubtitles dataset (Lison and Tiedemann, 2016) is a large subtitle-derived corpus from movies and television. The material is dominated by short dialogue lines and conversational phrasing, often formatted as rapid turn exchanges (Figure 2.4). Subtitles

provide broad coverage of everyday topics and interaction types, while remaining relatively "clean" compared to raw conversational transcripts.

OpenSubtitles excerpt

– But I don't want to go to summer camp.
– Are you kidding?
You'll have a great time.
Look at this one.

Figure 2.4: Representative excerpt from the OpenSubtitles dataset.

Simple Wikipedia Simple English Wikipedia[2] is an encyclopedia written in simplified English intended to improve accessibility. The text is encyclopedic and highly templatic, with frequent headings, short definitional statements, and repeated patterns across entries, as reflected in Figure 2.5.

Simple Wikipedia excerpt

=== Champmotteux ===
Champmotteux is a commune. It is in le-de-France in the Essonne department in north France.

=== Chatignonville ===
Chatignonville is a commune. It is in le-de-France in the Essonne department in north France.

Figure 2.5: Representative excerpt from Simple Wikipedia.

Switchboard Switchboard (Stolcke et al., 2000) is a corpus of spontaneous telephone conversations between speakers prompted by everyday topics. The Switchboard Dialog Act Corpus (SwDA) provides dialog-act annotations over a subset of calls and is widely used as a benchmark for conversational modeling. The data exhibit classic spoken fillers, as depicted in Figure 2.6.

2. https://simple.wikipedia.org/

Switchboard excerpt

A: Does it say something?
B: I think it usually does.
B: You might try, uh,
B: I don't know,
B: hold it down a little longer,
B: and see if it, uh,
A: Okay

Figure 2.6: Representative excerpt from the Switchboard dataset.

2.3 Evaluation

For evaluation we will again orient ourselves using the BabyLM shared task, as such small/under-trained models can not perform on most modern benchmarks which require basic instruction following. BabyLM employs a battery of different tasks which either measure model performance or human-likeness. Table 2.2 provides an overview of all the BabyLM language evaluation tasks as of 2025.

Table 2.2: Summary of BabyLM 2025 Evaluation tasks. The bolded tasks will be used in this thesis.

Task	**Metric Type**	**Fast Eval**	**Eval Mode**	**Scoring**
GLUE/SuperGLUE	Performance	–	Finetuning	Accuracy, F1
BLiMP	**Performance**		**Zero-shot**	**Accuracy**
EWoK	**Performance**		**Zero-shot**	**Accuracy**
Entity Tracking	**Performance**		**Zero-shot**	**Accuracy**
COMPS	**Performance**	–	**Zero-shot**	**Accuracy**
Reading Times	Human-Likeness		Zero-shot	Correlation
WUG_PAST	Human-Likeness	–	Zero-shot	Correlation
WUG_ADJ	Human-Likeness	–	Zero-shot	Correlation
Age of Acquisition	Human-Likeness	–	Checkpoints*	Correlation

*Requires model checkpoints at specified training intervals.

This work is primarily interested in performance-based evaluation. Most performance-based evaluations are based on the notion of perplexity, where the model assigns a perplexity score (corresponding to the likelihood of generation) to each member of a minimal pair. Here, we consider accuracy to be the proportion of pairs in which the member assigned a lower perplexity score is the expected preference. We omit the 'GLUE' task from our evaluations, as it consists of multiple sub-tasks that require fine-tuning, which would further necessitate hyperparameter tuning and occasional post-training on data comparable to our pre-training data. We will provide a short overview of the metrics and benchmarks used below:

Held-out Development Set Perplexity Perplexity is the standard metric used during language model training. It is supposed to represent how well models can predict language. It is essentially a length-normalized probability score for each sequence.

$$P\ w_1, w_2, \quad , w_n)^{\ \frac{1}{n}}$$

Perplexity represents the exponential of the average negative log-likelihood assigned to each token in a given sequence, with lower values corresponding to greater confidence (on average) in the model's next-token predictions. A model with perfect predictive performance would achieve a perplexity score of 1. Perplexity scores are often reported in their log form, also called the negative log likelihood (NLL).

$$\text{NLL} = \quad \frac{1}{n}\sum_{i=1}^{n} \log P\ x_i | x_{<i})$$

Benchmark of Linguistic Minimal Pairs (BLiMP) BLiMP (Warstadt, Parrish, et al., 2023) evaluates a model's adherence to various English grammar rules and conventions. BLiMP measures this adherence using minimal pairs. Each pair consists of two almost identical sentences where one is grammatically less correct than the other (we provide reference examples of these tasks in Table 2.5). The BLiMP tasks are split into 67 categories of measured phenomena, where each category has 1000 minimal pairs. The Supplement BLiMP is a supplementary BLiMP dataset created specifically for the BabyLM challenge with four additional syntactic/linguistic categories. We provide sample counts and category examples for this supplementary task in Table 2.4, and Table 2.5.

The used BabyLM evaluation set is subsampled to only include examples whose words also occur in the training corpus, and the BabyLM fast evaluation set which is used to assess intermediate checkpoints is further subsampled. Table 2.3 details the subsampled dataset makeup for both the filtered and fast BLiMP evaluation sets.

BLiMP is evaluated on whether the correct sentence in the minimal pair is assigned a lower (better) perplexity. The score is binary (0/1) and overall BLiMP accuracy is the average evaluation over all pairs. The reported human baseline agreement is 96.4%, which represents the percentage of the BLiMP minimal pairs where the human annotators' labels agreed with the labels assigned by the linguistic rules that they used to generate the data.

Table 2.3: BLiMP dataset statistics

Category	# Subtasks	BLiMP	BLiMP Filtered	BLiMP Eval
Anaphor Agreement	2	2000	1902	400
Determiner-Noun Agreement	8	8000	6793	1600
Subject-Verb Agreement	6	6000	5190	1200
Binding Principles	7	7000	6510	1400
Island Constraints	8	8000	7448	1600
Argument Structure	9	9000	7890	1800
Existential/Expletive Constructions	5	5000	4336	1000
NPI Licensing	7	7000	6261	1400
Wh-Constructions	7	7000	6179	1400
Irregular Morphology	2	2000	1903	400
Ellipsis	2	2000	1630	400
Quantifiers	2	2000	1965	400
Control/Raising	2	2000	1868	400
Total	**67**	**67000**	**59875**	**13400**

Table 2.4: BLiMP Supplement dataset statistics

Category	Supplement Filtered	Supplement Eval
hypernym	842	50
qa_congruence_easy	64	50
qa_congruence_tricky	165	50
subject_aux_inversion	3867	50
turn_taking	280	50
Total	**5218**	**250**

Table 2.5: Representative examples from BLiMP, differences are **bolded**

Category		Example Sentence
Anaphor Agreement	+	Katherine can't help **herself**.
	-	Katherine can't help **himself**.
Determiner-Noun Agreement	+	Raymond is selling this **sketch**.
	-	Raymond is selling this **sketches**.
Subject-Verb Agreement	+	Paula **references** Robert.
	-	Paula **reference** Robert.
Binding Principles	+	That actress who admired the cup praises **herself**.
	-	That actress who admired the cup praises **itself**.
Island Constraints	+	Who have those men revealed **they** helped?
	-	Who have those men revealed **who** helped?
Argument Structure	+	Diane **watched** Alan.
	-	Diane **screamed** Alan.
Existential/Expletive Constr.	+	There was **a** cat annoying Alice.
	-	There was **each** cat annoying Alice.
NPI Licensing	+	**Only** the sock that Larry admires ever warped.
	-	The sock that **only** Larry admires ever warped.
Wh-Constructions	+	Joel discovered **the vase that** Patricia might take.
	-	Joel discovered **what** Patricia might take **the vase**.
Irregular Morphology	+	Teresa **hid** every senator.
	-	Teresa **hidden** every senator.
Ellipsis	+	Brad passed one **big** museum and Eva passed several.
	-	Brad passed one museum and Eva passed several **big**.
Quantifiers	+	No actor complained about **more than** two actresses.
	-	No actor complained about **at least** two actresses.
Control/Raising	+	The restaurant was **easy** to drive to.
	-	The restaurant was **about** to drive to.
BLiMP Supplement		
Hypernym	+	If she has a dog, she must have a **mammal**.
	-	If she has a dog, she must have a **chihuahua**.
QA Congruence (Easy)	+	What did you get? I got a **chair**.
	-	What did you get? I got a **teacher**.
QA Congruence (Tricky)	+	Who cleaned? **David** cleaned.
	-	Who cleaned? **The patio** cleaned.
Subject-Aux Inversion	+	Is the novel he **is putting** away from the library?
	-	Is the novel he **putting away is** from the library?
Turn Taking	+	David: Should **you** quit? Sarah: No, I shouldn't.
	-	David: Should **she** quit? Sarah: No, I shouldn't.

Elements of World Knowledge (EWoK) EWoK (Ivanova et al., 2025) evaluates whether language models possess fundamental world knowledge necessary for building internal world models. It tests conceptual understanding across 11 knowledge domains (see Table 2.6) that cognitive science has identified as foundational for human cognition. These domains include knowledge that preverbal infants already possess, suggesting they are core components of intelligence. For every property EWoK has complementary context statements and contextually dependent conclusions. For example the complementary context statements could be: "Speaking up is hard for Chao" vs "Speaking up is easy for Chao" And the conclusions are: "Chao is shy" and "Chao is confident". More examples for all categories can be seen in Table 2.5.

EWoK basically does a double minimal pair comparison, where it tests whether $P\ T_1|C_1) > P\ T_1|C_2)$ and $P\ T_2|C_2) > P\ T_2|C_1)$. The model receives credit depending on how many conditions hold; it receives full credit (1 point) if both the conditions hold, partial credit (0.5 points) if only one condition holds and no credit if neither of the conditions hold. The total reported EWoK score is the average credit across all the items in the evaluation set, and a performance of 0.5 is representative of random-chance level accuracy.

Table 2.6: EWoK dataset statistics

Category	EWoK Filtered	EWoK Eval
agent-properties	2210	100
material-dynamics	770	100
material-properties	170	100
physical-dynamics	120	100
physical-interactions	556	100
physical-relations	818	100
quantitative-properties	314	100
social-interactions	294	100
social-properties	328	100
social-relations	1548	100
spatial-relations	490	100
Total	**7618**	**1100**

Table 2.7: Representative examples from EWoK

Category	Context and Target Sentences
agent-properties	*Context:* Yan successfully lied to Chao that the wheel was moved from the shop. + Chao **doubts** that the wheel is in the shop. - Chao **believes** that the wheel is in the shop.
material-dynamics	*Context:* Mohammed sees the **fork**. + Mohammed **taps** it. - Mohammed **stirs** it.
material-properties	*Context:* Jesse applied pressure to the balloon. It **remained intact**. + The balloon is **sturdy**. - The balloon is **fragile**.
physical-dynamics	*Context:* The football is moving **down**. + The football is **falling**. - The football is **rising**.
physical-interactions	*Context:* The **screwdriver** is exerting force upon the cooler. + The **screwdriver** is pushing the cooler. - The **cooler** is pushing the screwdriver.
physical-relations	*Context:* The train can only partially hide the boulder from view. + The **boulder** is bigger than the train. - The **train** is bigger than the boulder.
quantitative-properties	*Context:* There are **few** steaks relative to desks. + There are **fewer** steaks than desks. - There are **more** steaks than desks.
social-interactions	*Context:* **Yan** is showing Chao how to ski. + **Yan** is teaching Chao. - **Chao** is teaching Yan.
social-properties	*Context:* Speaking up is **hard** for Chao. + Chao is **shy**. - Chao is **confident**.
social-relations	*Context:* **Ali's** alumni spoke with Wei. + Ali is **Wei's teacher**. - Wei is **Ali's teacher**.
spatial-relations	*Context:* The shop is in front of Chao, and the zoo is behind. Chao turns **left**. + The shop is to the **right** of Chao. - The shop is to the **left** of Chao.

Entity Tracking Entity tracking (Kim and Schuster, 2023) tests whether language models can maintain and update representations of entity states as a discourse unfolds. This is a fundamental requirement for discourse understanding. The task probes whether models can track how operations change the state of entities.

The original task evaluated generation accuracy. For BabyLM 2025, the evaluation is adapted to a minimal-pair format where the model must assign higher probability to the correct continuation than to incorrect alternatives.

Table 2.8: Entity Tracking dataset statistics

Category	Entity Tracking Filtered	Entity Tracking Eval
regular	3152	3152
ambiref	3224	–
move_contents	3107	–
Total	**9483**	**3152**

Table 2.9: Representative examples from Entity Tracking

Variant	Prompt and Answer
regular	*Prompt:* Box 0 contains the wire, Box 1 contains nothing, Box 2 contains nothing, Box 3 contains the cigarette, Box 4 contains the card and the guitar, Box 5 contains the bowl, Box 6 contains the medicine and the pot. **Move the wire from Box 0 to Box 1.** Box 1 contains *nswer:* the wire.
ambiref	*Prompt:* Box 0 contains the small coffee and the big map, Box 1 contains nothing, Box 2 contains the **red cigarette** and the small drink, Box 3 contains the red disk, Box 4 contains nothing, Box 5 contains the **green cigarette** and the yellow car, Box 6 contains the big note. Move **the cigarette** from Box 5 to Box 1. Box 1 contains *nswer:* the green cigarette.
move_contents	*Prompt:* Box 0 contains the bone and the bus and the engine, Box 1 contains the newspaper, Box 2 contains nothing, Box 3 contains the ball and the picture, Box 4 contains the bread, Box 5 contains the block, Box 6 contains the tie. **Move the contents of Box 1 to Box 3.** Box 3 contains *nswer:* the ball and the newspaper and the picture.

COMPS COMPS (Misra et al., 2023) evaluates whether language models can attribute properties to everyday concepts and demonstrate property inheritance, the ability to infer that subordinate concepts inherit properties from their superordinates, see example 'comp_wugs' in Table 2.11. Accuracy is calculated over the minimal pairs and averaged.

Table 2.10: COMPS dataset statistics

Variant	Subtype	Examples
comps_base	taxonomic	12335
	co-occurrence	12335
	overlap	12335
	random	12335
comps_wugs	undistracted	13896
comps_wugs_dist	distraction before	13896
	distraction in-between	13896
Total		**91028**

Table 2.11: Representative examples from COMPS.

Variant	Sentence Pair
comps_base (taxonomic)	*Property:* can bark + A **dog** can bark. - A **fox** can bark.
comps_wugs	*Property:* can Fly + A wug is a **robin**. Therefore, a wug can fly. - A wug is a **penguin**. Therefore, a wug can fly.
comps_wugs_dist	*Property:* can Fly + A wug is a robin. A dax is a dog. Therefore, a **wug** can fly. - A wug is a robin. A dax is a dog. Therefore, a **dax** can fly.

2.4 Language Model Foundations

This section will introduce the basics of language modeling as necessary to understand this thesis. It will begin with a short introduction of pre-training language modeling objectives and then give a short overview of the transformer architecture (Vaswani et al., 2017) and optimization methods used for machine learning models, including the AdamW optimizer that we use for our experimental setup.

2.4.1 Pre-Training Objective

Modern language models are usually trained in two stages: i) the 'pre-training' stage, where large diverse language model corpora are used to train the model in an unsupervised fashion for general language understanding, and ii) an ensemble of post-training methods, designed to adjust the probabilities such that the language generated is helpful and coherent in the target application. As this thesis is only concerned with pre-training, we will discuss some of the most common pre-training methods.

Pre-training is considered unsupervised, as the training objective relies only on the raw text and does not require human annotation or any kind of manual annotation of the data to ensure it has a supervised learning signal. The basic structure for pre-training is for the model to learn the underlying statistical structure of a language corpus, which is typically posed as a problem where some parts of the input text are masked or removed, and the model is then tasked to "reconstruct" these masked out portions of text. In the following section I will introduce relevant pre-training methods for language models.

Causal Language Modeling

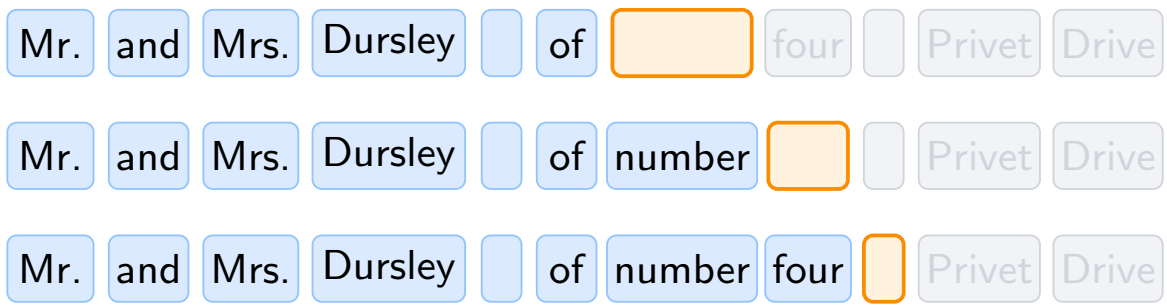

Figure 2.7: A visualisation of the information each CLM prediction has.

The dominant objective for generative models is called Causal Language Modeling (CLM) (Vaswani et al., 2017). Here the language model is prompted to predict the next token given previous context:

$$P\ x_i|x_{<i})$$

This has the benefit that it mirrors human language generation. In transformer architectures this is generally implemented such that given a sequence of words, the model predicts the next word probability for all words considering only their past tokens.

Masked Language Modelling

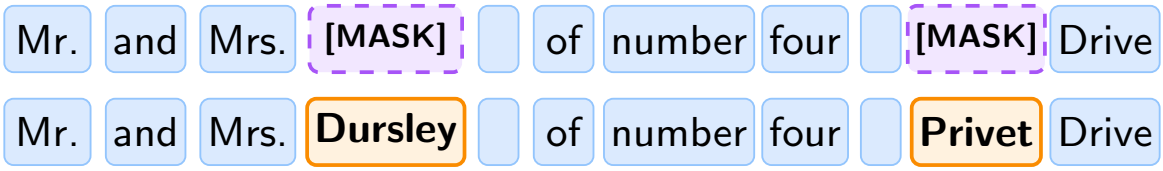

Figure 2.8: A visualisation of the information each MLM prediction has.

Masked Language Modeling (MLM) is the training strategy used in encoder models such as BERT (Devlin et al., 2019) and RoBERTa (Liu et al., 2019). The basic form is that the model should predict a token, or a set of tokens, given all surrounding tokens:

$$P\ x_i|x_{\neq i})$$

This is implemented using a mask, so the sequence is fed into the model with the location of the tokens to be predicted masked. Usually around 15% of tokens are masked.

Permutation Language Modeling (PLM)

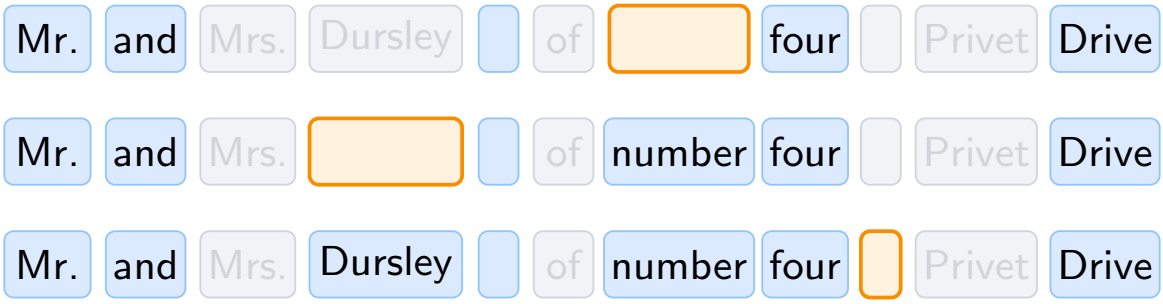

Figure 2.9: A visualisation of the information each PLM prediction has.

Permutation Language Modeling (PLM) is similar to CLM in that there are no mask tokens introduced and it can be run over the whole sequence at once. It is effectively the same as CLM, only that the context each token sees is not necessarily the past tokens in order, but rather the past tokens of a permuted sequence. It can also be understood similarly to MLM models if, for instance, training is only done over the last 15% of the tokens, as is standard for XLNet (Z Yang et al., 2019), the most popular PLM-trained model.

$$P\ x_{z_i}|x_{z_{<i}})$$

2.4.2 Transformer rchitecture

This thesis will focus solely on transformer LLM architectures. For this reason, we will provide a short overview of how a transformer model functions and is structured. Since there are a large

variety of transformer implementations, we will largely focus on the Llama 3 implementation of the transformer architecture, and explain Grouped Query Attention as well as the positional embedding RoPE (Su et al., 2024).

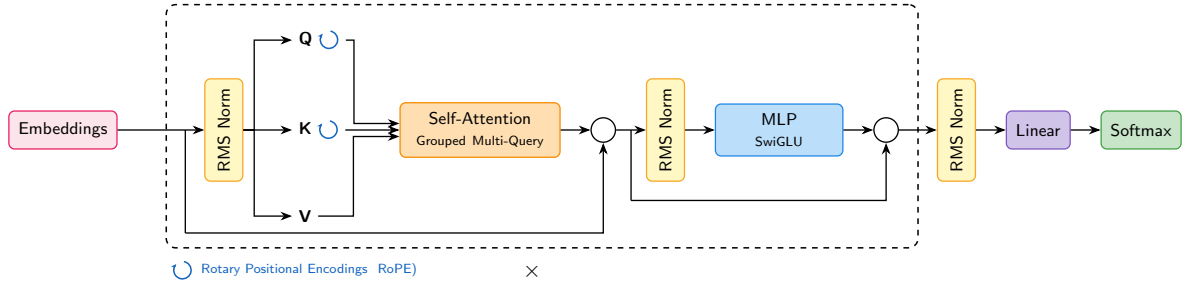

Figure 2.10: A visualisation of the Llama architecture.

Tokenizer and Embeddings

In language models, sequences are typically fed into models by splitting them up into tokens. Tokens can be considered the language model version of words. Since a language model has a limited vocabulary that cannot include all words or word misspellings, a tokenizer is trained to build sensible word segments. The tokenizer in our experiments has ≈ 16000 tokens. The first step of the model is to transform these tokens into vectors with which the model can work internally. This is done via the Embedding layer, which is essentially a mapping from token to a trained vector.

Layer Structure

After the Embedding step, the vector representations run through multiple layers, which form the core of the model. These layers are generally made up of a self-attention module where this representation interacts with the other token representations, followed by a Multi-Layer Perceptron (MLP) layer.

Self ttention

Self-attention is the only portion of the transformer where information is exchanged between different tokens. The first step is to transform the current representation into key, query, and value representations. The current token will 'attend' to other tokens in the input. During training of a transformer, all tokens are passed into the model simultaneously. Depending on the training objective, this attention is allowed to see only a subset of the other tokens.

Each token's query vector will calculate this similarity to all visible token key vectors via the dot product. Each token then has a similarity distribution over visible tokens, this distribution is sharpened using the softmax and the corresponding value vectors are retrieved. It has been shown that it is more effective to not do one query per layer, but instead perform multiple queries with lower dimensionality. The original transformer implemented so-called 'multi-head

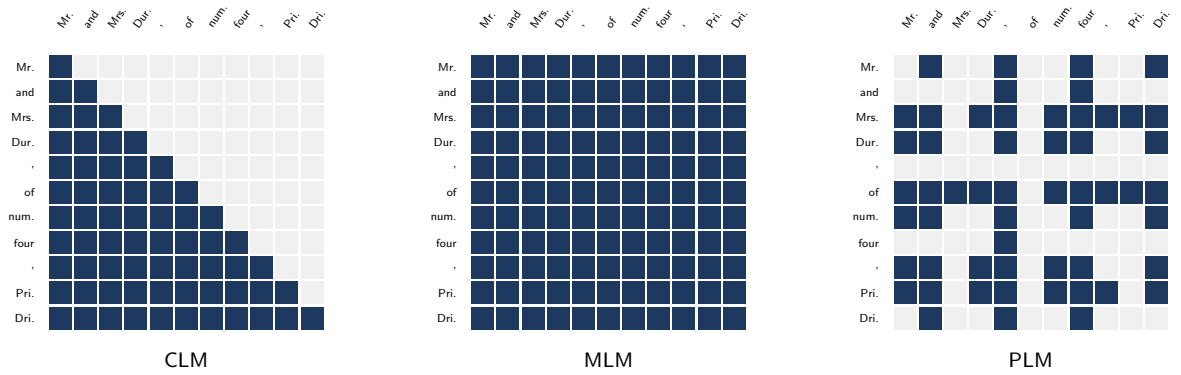

Figure 2.11: A visualisation of masks in different training objectives.

attention': here the incoming token representation is projected into h smaller key, query, and value vectors. After attention is performed, the resulting retrieved values are concatenated, resulting in a vector of the original token dimension.

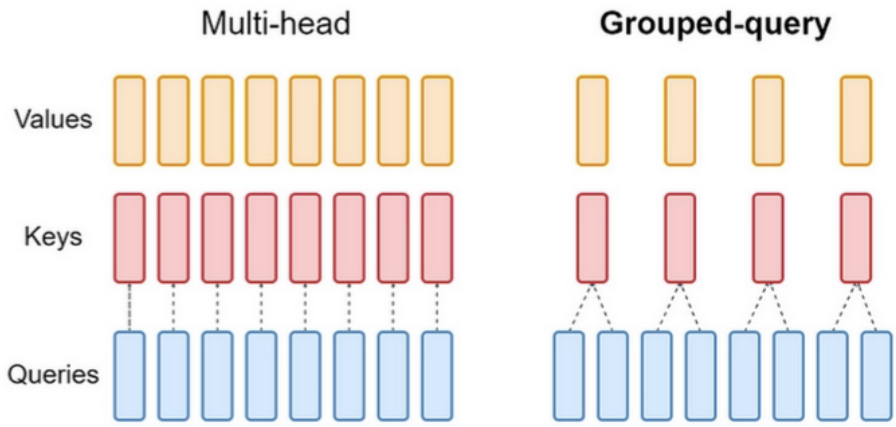

Figure 2.12: A visualisation of Grouped Query Attention (Ainslie et al., 2023).

A more efficient implementation is the Grouped Query Attention (GQA), where some heads share the same keys and values, reducing the number of training parameters necessary for key/value projection. While this introduces a slight reduction in model capacity, the more important motivation is that it significantly reduces the size of the KV cache. The KV cache is largely inconsequential for this work, but in causal language models during generation, naively one would pass all previous tokens in for each new token generation and redo all their calculations. However, since self-attention is the only module where tokens influence each other, and this influence is unidirectional, the most efficient approach is to cache past key and value states for reuse during future token generation. Reducing the number of these states significantly lowers inference costs.

Rotational Positional Embeddings

Within transformers there is no inherent knowledge of token ordering. This is most extreme in MLM models, where every token can attend to every other token per self-attention layer, yet

intrinsically there is no information as to what order these other tokens are in. The original transformer architecture added position-dependent vectors to the initial token embeddings, hoping the model would disentangle this information independently. However, researchers have found that models are relatively poor at this and that this approach develops position-specific oddities.

The Llama models and many other prominent models released in recent history generally make use of Rotary Positional Embeddings (RoPE) (Su et al., 2024). The idea behind positional embeddings is that the model should consider the similarity of two tokens depending only on their distance, not their absolute position in the text. For example, among themselves the tokens "Mr. Dursley and Mrs. Dursley" should internally have the same attention score regardless of whether they appear at the beginning or end of a sequence fed to the model. To achieve this, positional embeddings are now added in all self-attention modules, not at the initial embedding stage.

RoPE uses the concept of rotational invariance for this: if we rotate vectors according to their position in a sentence, the relative rotation between vectors depends on their distance and stays consistent regardless of where in the input they are located. Rotations are well defined in two dimensions, so before passing the key and query vectors into self-attention, they are altered. Each query and each key vector is rotated based on their position in the sequence:

$$\begin{aligned} qk_{mn} &= R_{\theta,m} W_q x_m)^T \; R_{\theta,n} W_k x_n) \\ qk_{mn} &= x_m^T W_q^T R_{\theta,m}^T R_{\theta,n} W_k x_n \\ qk_{mn} &= x_m^T W_q^T R_{\theta,n \;\; m} W_k x_n \end{aligned}$$

Since rotation in two dimensions is well defined, the RoPE paper rotates dimensions in pairs of two, using different frequencies for different pairs (Su et al., 2024).

2.4.3 Optimization

Stochastic Gradient Descent (SGD) is the base form of LLM optimization. Since large language models are trained over so much data, the loss cant be calculated over the whole dataset, but instead the dataset is subsampled and the gradient is an approximation given $|seqence| \times \#batches$ tokens. In SGD the weights are then subtracted by $lr \times g$.

$$w_{n+1} \leftarrow w_n \quad lr \times g_{n+1}$$

SGD with momentum is a popular continuation of the approach. Where the gradient is a moving average of the current and past gradients:

$$m_{n+1} \leftarrow \quad m_n + \; 1 \quad \tau) g_{n+1}$$

$$w_{n+1} \leftarrow w_n \quad lr \times m_{n+1}$$

So SGD can accumulate directions, leading to an amplification of consistent signal. In the special case where $\quad = \tau$ momentum turns into an exponential moving average, so here no signal get accumulated, its a smoothing factor over past gradients, leading to less noisy gradient updates.

RMSProp is another popular continuation of SGD. Here, the optimizer maintains a running average of the squared per-parameter gradients, and each update is normalized by the root of this average. What this does is that it changes the per-parameter learning rate and normalizes it so as to ensure that parameters with consistently large gradients receive smaller updates, whereas parameters with small gradients would conversely receive larger updates. This ensures that significant differences in the gradient initialization does not cause significantly large differences in the optimization trajectory. The directionality of the update still follows the sign of the gradient, with only the magnitude being normalized.

AdamW (Loshchilov and Hutter, 2019) is one of the most ubiquitous optimizers used for LLM training. While it is often described as a combination of momentum and RMSProp, its function is much closer to RMSProp than momentum. AdamW also normalizes per-parameter gradient size.

AdamW can be understood as a signal-to-noise ratio, where it accumulates the mean gradient direction (momentum) and compares it to the mean squared gradient magnitude (second moment) as shown in the equations below.

$$m_t = \quad {}_1 m_t \quad {}_1 + \quad 1 \qquad {}_1) g_t \tag{2.1}$$

$$v_t = \quad {}_2 v_t \quad {}_1 + \quad 1 \qquad {}_2) g_t^2 \tag{2.2}$$

$$\hat{m}_t = \frac{m_t}{1 \qquad {}^t_1} \tag{2.3}$$

$$\hat{v}_t = \frac{v_t}{1 \qquad {}^t_2} \tag{2.4}$$

$$\tag{2.5}$$

When the gradients are observed to be noisy and inconsistent, the momentum is smaller compared to the average gradient magnitude, and so the update to the gradients is generally smaller and more conservative. On the other hand, when the gradients are consistent and aligned, the momentum is closer to the gradient magnitude, and the update is roughly equal to the learning rate.

$$\theta_t = \theta_t \quad {}_1 \quad \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon} \quad \eta \lambda \theta_t \quad {}_1 \quad \text{(AdamW update)} \tag{2.6}$$

Here η denotes the learning rate, λ is the value of the weight decay and ε signifies a small constant that prevents any division by zero.

3 Epoch Scaling

3.1 Introduction

Prior research on training highly performant models through hyperparameter optimization often focuses on optimizing the learning rate. However, previous work of the BabyLM challenge (Tastet and Timiryasov, 2024) and a recent scaling paper (Kim et al., 2026) have demonstrated the importance of weight decay when it comes to managing epoched training regularisation. Importantly, epoched regimes can benefit from weight decay values that can be as far as up to $30\times$ higher than the standard value of 0.1 typically used in training configurations.

In line with this, Section 3.3 runs a weight decay and learning rate hyperparameter analysis over three different scales of models, and finds that the optimal hyperparameters all lie on an epoch size dependent $lr \times wd$ line.

The epoched models show a sawtooth-like epoch-aligned dev loss pattern, with the optimal loss at the end of each epoch. This caused the optimal checkpoint at the end of every run to fall on an epoch boundary. In section 3.4, we describe the mechanism for this and show that, despite this random effect, the random reshuffling strategy maintains the optimal ordering for training. We further show that the variation in the dev loss depends on the average distance to a sample's prior appearance within each batch, with early batches correlating much more strongly with the running momentum than later ones. This effectively causes a per-epoch learning rate schedule over the course of training.

Section 3.5.1 mitigates model overfitting using a SWA-like EMA over the run weights, and a constant learning rate. This setup allows the evaluation of arbitrary epoch lengths in a single run. We show that by training the model with this strategy, the optimum dev loss stays stable according to the $wd \times lr$ product, regardless of the wd/lr values chosen. We hypothesize that this is because the effective learning rate is the same and stabilizes quickly for normalized matrices when using weight decay. We show that the EMA models exhibit a lower overall sharpness and outperform all cosine-trained models trained prior. Based on the static performance-focused metrics we evaluate, the best tuned EMA models performed second among all 121 2026 BabyLM challenge models.

3.2 Related Works

3.2.1 Epoched LLM training

BabyLM Papers The BabyLM challenge provided examples from which the importance of weight decay could be anticipated. One of the most compute efficient papers in the 2024 BabyLM shared tasks was the BabyLlama2 (Tastet and Timiryasov, 2024) paper. This paper trained two teachers and then used their logit distribution to distill a student model. While training the teachers, they performed a coarse-grained hyperparameter scan, treating each

parameter independently. During this, they found an optimal weight decay value of 5, 50-500 times larger than typical for non-epoched language pretraining. However the paper does not comment on this result. The BERT based approach in Samuel et al. (2023) also notes that weight decay tuning was very important, and they increased the weight decay from 0.01 to 0.1.

Scaling Data-Constrained Language Models While this section's experiments were created (Muennighoff et al., 2025) was the most up to date paper on model performance and model scaling impact when training under repeated data. The paper found that epoching was fine for a small number of epochs and at that point model performance would decrease. The paper also found that there was a limit to how much model size could improve behavior. Notably the paper explored all of this using hyperparameters discovered in non-epoched settings.

Pretraining under infinite compute During the time this section's experiments were run, related work in this area (Kim et al., 2026) was released. It studied two strategies for improving model performance given a finite dataset, of which one was model scaling while hyperparameter tuning weight decay. The other strategy involved ensembling models, i.e. averaging the probability distribution of multiple models run on the same dataset. This paper showed that optimal weight decay is significantly higher when working with epoched datasets, and furthermore, they proposed it was related to the $\frac{\text{token}}{\text{model parameter}}$ ratio. Contrary to Muennighoff et al. (2025), they found that scaling model size continued to improve performance, although with diminishing returns.

3.2.2 Random Reshuffling and Sawtooth Dynamics

Random Reshuffling and Data Ordering Random reshuffling (RR) draws fresh permutations for each epoch during training. It is generally considered the best randomized scheme during epoched training. Prior research has explored saved gradient based more effective random orderings, such as Lu et al. (2023), who show that herding-based permutations have a better convergence rate on smooth non-convex objectives. We tested their scheduler in Section 3.4; however we do not get better results, and chose not to pursue it further.

Epoch- ligned Loss Oscillations Liu and Ma (2025) describe the *Epochal Sawtooth Phenomenon*, a phenomenon they find in the training loss curvature. The training loss drops sharply at the beginning of each epoch and then rises gradually, producing an epoch-aligned sawtooth. We find a similar training loss effect when doing what this paper did when approximating training loss using the current epoch batch. However this introduces a confound in our setting. When overfitting, each sample becomes significantly less perplexing for a longer and longer period of time, such that if evaluated again in a following epoch, depending on where in the epoch it has been evaluated, its loss will still be comparatively lower or higher. Liu and Ma (2025) attribute this effect to the second moment estimate in combination with very low sample training distances at epoch edges.

3.2.3 Weight Decay Dynamics

damW as an Exponential Moving verage Wang and Aitchison (2025) show that the weights learned by AdamW can be interpreted as an exponential moving average of recent updates, whose key hyperparameter is the EMA timescale $\tau = 1/\ \eta\lambda)$, or $\tau_{\text{epoch}} = 1/\ \eta\lambda\, N/B)$ when expressed in epochs. Empirically, the optimal timescale measured in epochs is roughly constant across model and dataset sizes, which implies a one-to-one trade-off between learning rate and weight decay at the optimum. This view predicts the lr $\times$ wd line of optima we observe in Section 3.3.2.

Rotational Equilibrium Kosson et al. (2024) provide a mechanistic account of how learning rate and weight decay create the same model dynamics. For (approximately) scale-invariant weights, the interplay of weight decay and the gradient update drives training into a *rotational equilibrium* in which the expected angular update per step is constant and determined by $\eta\lambda$. This positions weight decay not as a decaying force reducing function space, but rather a effective learning rate size control impacting affecting regularisation through this lever. We evaluated and found rotational equilibria in Section 3.5.1 due to this paper.

3.2.4 Weight veraging and Schedulers

Stochastic Weight veraging Izmailov et al. (2018) introduce Stochastic Weight Averaging (SWA), a method averaging weighted during an SGD run, they show that averaging weights reduces run noise and reveals a flat minima, and better test performance. Our run-weight EMA is the exponentially weighted analogue of this procedure, and the reduced sharpness of our averaged models (Section 3.5) matches the flatness mechanism they propose.

veraging as a Substitute for Learning Rate Decay Hägele et al. (2024) shows that a windowed SWA over constant learning rate training runs can approximate a cooldown phase. They show that actual cooldown still results in better models. Our setup adopts this constant-rate-plus-averaging paradigm in the epoched regime, as to also avoid multiple cosine decay predetermined runs.

3.3 Scaling Hyperparameter Sweep

This section describes a hyperparameter sweep over learning rate and weight decay, where each model is trained on a total of 100M tokens. Each of these models belong to one of five epoching regimes– 6.25M tokens $\times$ 16, 12.5M tokens $\times$ 8, 25M tokens $\times$ 4, 50M tokens $\times$ 2 and 100M tokens $\times$ 1. Evaluations are performed over three model sizes of 48M, 99M and 286M parameters respectively.

3.3.1 Methodology

Stable Hyperparameter The models are trained on sequences of 128 tokens, following the example of BabyLlama2 (Tastet and Timiryasov, 2024). Default Adam optimizer values as well as weight initialization magnitudes were chosen. Both the initialization of the model and the data iteration was fixed using a seed. We chose the 'ltg/gpt-bert-babylm-base' tokenizer trained in tokens.

Table 3.1: Training Settings

Parameter	**Value**
Sequence Length	128
Optimizer	AdamW
Adam $_1$	0.9
Adam $_2$	0.999
Adam ε	10 ^{8}
Seed	42
Dev Eval Tokens	100K
Vocab Size	16384
Tokenizer	ltg/gpt-bert-babylm-base
Weight Initialization	Uniform, range $\pm$0 02
Precision	FP16
Batch Size	128
Warmup Steps	600
Total Training Tokens	100M
Learning Rate Schedule	Warmup + Cosine Decay

Hardware The hyperparameter search was primarily performed using a grid of NVIDIA 1080 Ti GPUs which have 11GB of VRAM memory. Due to size limitations, models on these GPUs were trained on batches of size 32 with a gradient accumulation of 4 ($4\times32 = 128$) to assure equal effective batch size. The 286M parameter models were trained on an NVIDIA A100. Overview can be seen in Table 3.2.

Table 3.2: Implementation Details

Parameter	Value
Hardware (48M, 99M)	NVIDIA 1080 Ti
Hardware (286M)	NVIDIA A100

Model rchitecture We opted for models in the toy model regime, and kept layer counts static, as width variation has been associated with improvements in monotonic perplexity (G Yang et al., 2022). The KV to head ratio was chosen to stay fixed at 1/4. The intermediate size ratio is typically 2 68 $\times d$ (where d is the size of the hidden state dimension) for SwiGLU architectures, rounded off to the nearest multiple of 256. Following convention, we set head count and head dimensions such that they correspond to the hidden size when multiplied.

Table 3.3: Model Architecture Specifications

Specification	48M	99M	286M
Hidden Size (d)	512	768	1280
Intermediate Size	1536	2048	3584
Num Layers (L)	16	16	16
Attention Heads	8	12	20
KV Heads (GQA)	2	3	5
Head Dimension	64	64	64
Vocab Size	16384	16384	16384

The models are named after the number of non-embedding parameters they possess, as seen in Table 3.4.

Table 3.4: Model Parameter Counts

Parameters	48M	99M	286M
Embedding	16.78M	25.17M	41.94M
Non-Embedding	48.25M	99.12M	285.78M
Total	65.03M	124.28M	327.72M

Hyperparameter Search Strategy Hyperparameter searches were first approximated using a rough grid search over learning rates (η) $[3\times10^{-4}, 5\times10^{-4}, 7\times10^{-4}, 1\times10^{-3}, 1\,4\times10^{-3}]$ and weight decay (λ) $[0\ 1, 0\ 5, 1\ 0, 2\ 5, 5\ 0, 7\ 5, 10\ 0]$. As a second step, optimal parameters were further approximated using a coordinate descent like algorithms (see the Appendix, Algorithm 1). Given the current optimum, we would look at the surrounding points, and check which one was closest, and this would define the current radius. It would then attempt to train models in all four directions $+\eta, \quad \eta, +\lambda, \quad \lambda$. If they all already exist, the radius would be halved. This would repeat until the radius was smaller than $\tau\times$ the current optimal values.

During adaptive search, the convergence threshold $\tau = 1\ 1$ and maximum iterations limit $T = 20$ were used. The maximum iteration limit was never reached.

Training Dynamics Metrics To analyze learning dynamics across model components, we optionally log detailed metrics regarding the models current parameters. This logging occurred every 10 training steps. We logged standard values, such as the average per token NLL of the current batch, the current learning rate, or the norm of the raw gradient. More detailed metric logging is described in Appendix A.1.

Dataset Construction We subsample the dataset described in Section 2.2. When preparing the dataset, each dataset was split into subdocuments and this was done at natural boundaries where possible. We provide the segmentation details below:

- Guttenberg: new book, marked by '= = ='
- Simple Wikipedia: new article, marked by '= = ='
- CHILDES/BNC/Subtitles: new conversation/scene, marked by an empty line
- Switchboard: every 200 lines as there was no identifiable splits

These subdocuments were then shuffled, tokenized using the 'ltg/gpt-bert-babylm-base' tokenizer and saved. The same process was applied to the separate development dataset corpora files. All training dataset splits were generated by selecting the first n tokens from the large training file. This way, all smaller datasets are a subset of the larger ones.

All datasets were epoched such that the total token count always equaled 100M tokens, since all runs had the same batch size of 128, this lead to identical learning rates schedules across training runs. The word count reported in Table 3.5 is based on an estimate of 1.764 tokens/word, based on the total 100M word dataset statistic seen in Table 2.1.

Table 3.5: Data Conditions

Unique Tokens	Unique Words	Epochs
6.25M	3.54M	16
12.5M	7.09M	8
25M	14.18M	4
50M	28.35M	2
100M	56.70M	1

An overview of all dataset sizes can be found in the Appendix, Table A.5 .

3.3.2 Results

The results of the hyperparameter search can be seen in Figure 3.2. Here we can see a visualization of the optimal development set loss per configuration. Learning rate generally falls as model size increases, while weight decay correspondingly increases. Weight decay generally increases as data becomes less unique and repetitions increase. Notably in low data regimes, it looks as though the optimal weight decay is static for multiple learning rate values, as it forms a horizontally elongated area of optimality.

Weight Decay Tuning Enables Continued Scaling

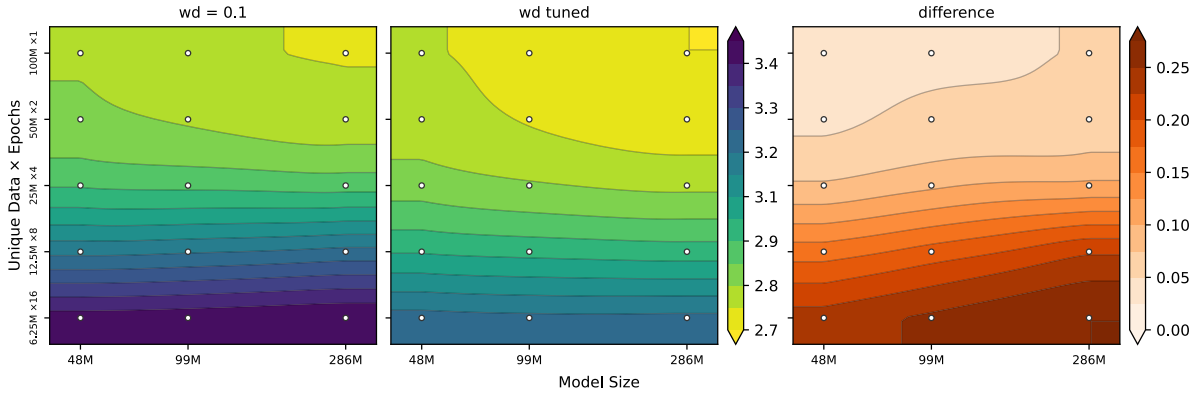

Figure 3.1: A contour plot visualization of the best dev loss over model and unique data size. The first two plots are a contrast the conventional wd=0.1 and wd tuned settings. The rightmost plot is a direct loss difference.

We see **significant improvements when tuning the weight decay parameter** in addition to learning rate parameter, as already shown in Kim et al. (2026). These improvements are visualized in Figure 3.1. Weight decay tuning brings the most significant improvements as data repetition, and model size increases.

Notably **scaling weight decay allows us to scale model size indefinitely** for all repetition magnitudes, as already found in Kim et al. (2026). This can be seen in Table 3.6, where all tuned wd models are optimal at the largest model size. While the standard wd=0.1 models mirror the results from Muennighoff et al. (2025), in the sense that data-scaling up to four epochs is tolerable.

Table 3.6 also shows that **optimal** $lr \times wd$ **stays consistent per dataset size**, with larger models leaning towards larger learning rates. This dependence mirrors what Wang and Aitchison (2025) found for μp-scale invariant runs. A further discussion will follow in Section 3.3.2.

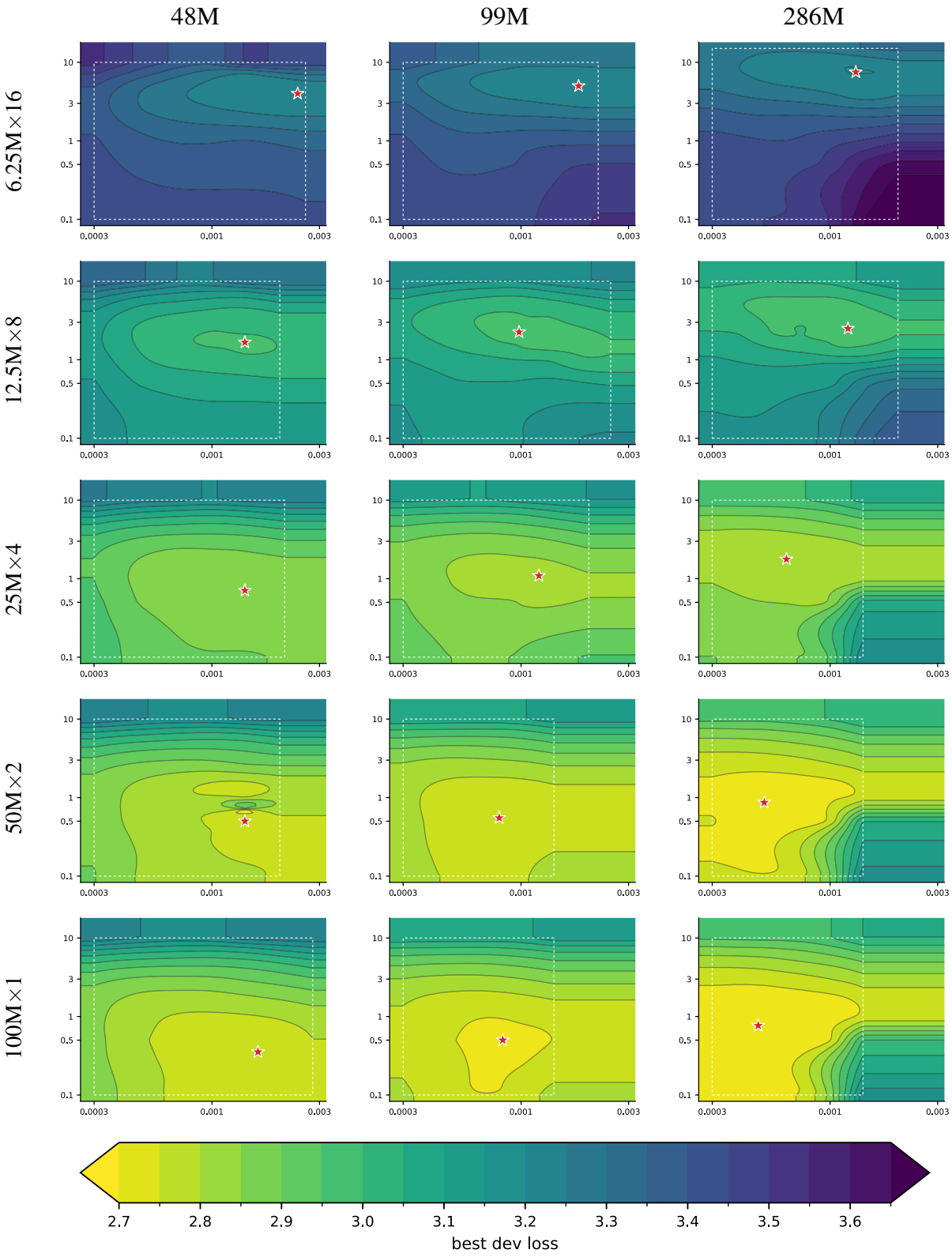

Figure 3.2: Best-dev-loss surfaces over learning rate (x log) and weight decay (y log) for every model size (columns) and data condition (rows). All fifteen panels span identical axis ranges and share one color scale. The white rectangle corresponds to the region actually measured.

Table 3.6: Optimal hyperparameters and performance for each model × dataset combination. Reported for the learning rate (η) and the weight decay (λ). $\text{Dev}_{0\ 1}$ corresponds to a weight decay of 0.1

Data	**Model**	η^*	λ^*	$\eta^* \times \lambda^*$	**Dev Loss**	**Dev**$_{0\ 1}$
6.25M×16	48M	2.40e-3	4.0	9.60e-3	3.238	**3.439**
6.25M×16	99M	1.80e-3	5.0	9.00e-3	3.233	3.448
6.25M×16	286M	1.30e-3	7.5	9.75e-3	**3.230**	3.468
12.5M×8	48M	1.40e-3	1.7	2.38e-3	3.030	**3.165**
12.5M×8	99M	9.80e-4	2.2	2.16e-3	3.015	3.174
12.5M×8	286M	1.20e-3	2.5	3.00e-3	**3.007**	3.199
25M×4	48M	1.40e-3	0.7	9.80e-4	2.897	2.945
25M×4	99M	1.20e-3	1.1	1.32e-3	2.872	2.937
25M×4	286M	6.40e-4	1.8	1.15e-3	**2.848**	**2.928**
50M×2	48M	1.40e-3	0.5	7.00e-4	2.832	2.849
50M×2	99M	8.00e-4	0.6	4.80e-4	2.807	2.829
50M×2	286M	5.10e-4	0.9	4.59e-4	**2.767**	**2.802**
100M×1	48M	1.60e-3	0.4	6.40e-4	2.815	2.825
100M×1	99M	8.30e-4	0.5	4.15e-4	2.787	2.799
100M×1	286M	4.80e-4	0.8	3.84e-4	**2.749**	**2.771**

Optimal Hyperparameter Invariance

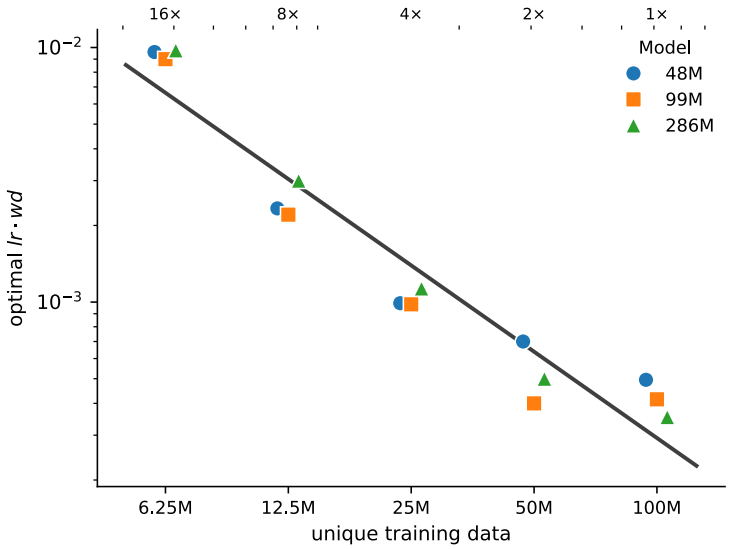

Figure 3.3: Optimal $lr \cdot wd$ against epoch length for optimal hyperparameters. The multipliers above remind of the epoch repetitions.

All optimal $lr \times wd$ parameters lie on a line in log-log space. This line has a slope of 1 13 ($R^2 = 0\ 92$). The models land on the same product within each conditions and across conditions and follow the Wang and Aitchison (2025) predicted wd scaling behavior for AdamW, in that it predicts that $lr \times wd \times$ "steps per epoch" will stay consistent across runs. The rule applies generally, although it is not clear to what extent the deviations from the line are because we do not use μ-p scaling.

Wang and Aitchison (2025) tested their scaling behavior mainly on vision models, and with static epoch counts and epoch length variation. Our results indicate that the law spans across different epoch counts, which is something they never tested.

It stands to reason that this result can be used by practitioners to tune hyperparameters more efficiently, by extrapolating from a few to many epochs and from smaller to larger models.

3.4 Epoch- ligned Loss Oscillation under Reshuffling

When analysing the hyperparameter sweeps optima, an interesting trend emerged. The positions of an optima is always located at the end of an epoch, never in between. The epochs at which an optimum is reached are also highly clustered. Runs in the high data regime end their run after the final epoch (step 6100) while the low data regimes (12.5M and 6.25M) finish their runs at the respective second to last epoch.

The models' dev evaluation values go through a cyclic sawtooth pattern with the lowest point located right in between epochs. This phenomenon can be seen in Figure 3.4. This section will analyze what causes this pattern, and how it impacts training.

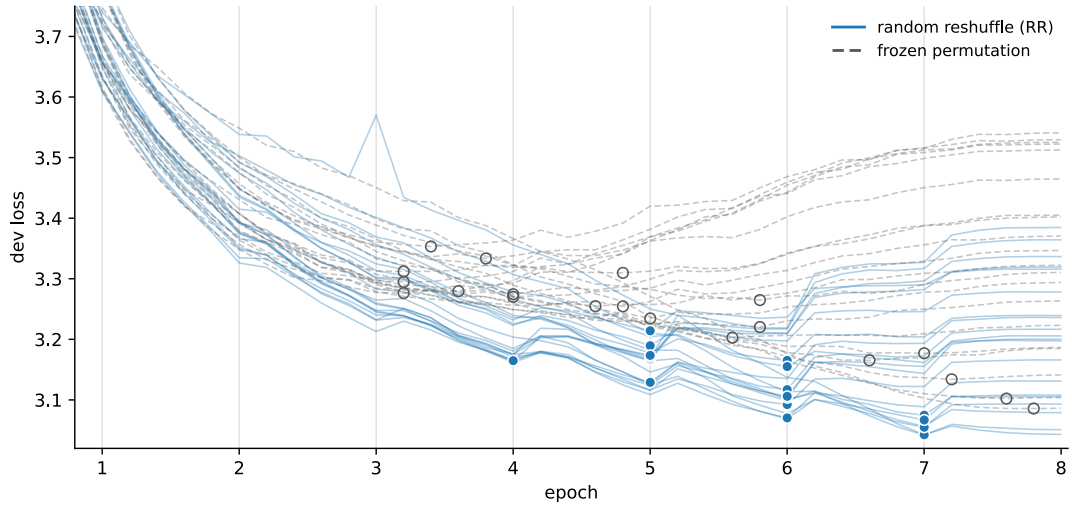

Figure 3.4: All 40 runs of the matched set (reshuffle solid, frozen dashed) with a marker at each run's dev-loss argmin. Reshuffled argmins land only on integer epoch boundaries (filled columns at epochs 4–7); frozen argmins (open) drift continuously between epochs 3 2 and 7 8. In every one of the 20 matched pairs the frozen minimum lies 0 043–0 153 above its reshuffled twin.

3.4.1 Methodology

Ordering control To isolate the cause for the sawtooth pattern we ran 20 of the 48M 12.5M $\times$ 8 hyperparameter tuning runs using the same permutation (shuffle once/frozen permutation) each epoch. Training settings remained otherwise the same as in the original hyperparameter runs, see Section 3.3.1. The grid is made up of $lr \in \{3\times10^{-4}, 5\times10^{-4}, 7\times10^{-4}, 1\times10^{-3}, 1\,4\times10^{-3}\}$ and $wd \in \{0\,1, 0\,5, 1\,0, 2\,5\}$. Every run has a corresponding random reshuffle (RR) twin already run as part of the hyperparameter tuning, differing only in data ordering.

Toy testbed. To be more efficient, separate data order experiments were run on a 4.2M model using the toy task of learning TinyShakespeare. For details regarding training see Table 3.8, for TinyShakespeare see Table 3.7.

Table 3.7: Toy-testbed dataset.

Corpus	TinyShakespeare
Characters	1,115,394
Split	90/10 train/dev
Vocabulary	65 (characters)
Steps per epoch	61

Table 3.8: Toy-testbed model and training configuration.

Architecture	Llama
Layers	4
Hidden size	256
Heads	4
Intermediate size	1024
Parameters	4.2M
Sequence length	256
Optimizer	AdamW
$_1$”	0.9
$_2$	0.95
Schedule	cosine to 0 1$\times$
Warmup	100 steps
Batch size	64
LR	$3 \times 10^{\ 4}$
WD	0.1
Epochs	30

Ordering regimes. We tested six orderings, as described in Table 3.9, each probing one aspect of the reshuffling advantage.

Table 3.9: Data-ordering regimes presented in this section, and what each isolates.

Regime	Definition	Probes
Random reshuffle (RR)	new random order every epoch	standard practice
Frozen permutation	shuffled once, replayed every epoch	no order variation
Alternate-reversed	every second epoch replays the previous order backwards	the revisit-distance jump at the epoch crossing
Static batches	random reshuffle with static batches	batch randomness importance
Minimum-gap shuffle	RR with a min distance between samples	short min revisit distances
I.i.d. w/ replacement	batches sampled with replacement	the epoch structure itself

Metrics Logged During the analysis of the different data-ordering schemes a detailed snapshot is taken every four steps. Logging includes per parameter group (embeddings, attention matrices, layer norms, MLP, output head) mean absolute gradient, the AdamW optimizer's first and second moments $|\hat{m}|$ and $\sqrt{\hat{v}}$.

3.4.2 Results

We verified that the sawtooth effect was caused by random per epoch shuffling, by running 20 runs of the 12.5M $\times$ 8 hyperparameter grid runs using the "shuffle once" permutation. The results can be seen seen in Figure 3.4, where the blue random reshuffle runs reach consistently better and lower results This effect can be very clearly see in Table 3.10, where RR runs have their optimum located at the intersections 100% of the time, while the frozen shuffle alternative has its optimum spread according to the expected 20% (runs are evaluated 5 times within an epoch).

Table 3.10: Position of the best-dev-loss checkpoint relative to epoch boundaries (12.5M$\times$8, 48M, cosine, 100M tokens).

Data ordering	Runs	On boundary	Chance
Random reshuffle (matched cells)	20	100 %	20 %
Frozen permutation (matched cells)	20	20 %	20 %

What causes the pattern?

To analyze what causes the pattern and possibly improve performance by reducing it, this section tested a variety of sampling algorithms, as described in the methodology this is done

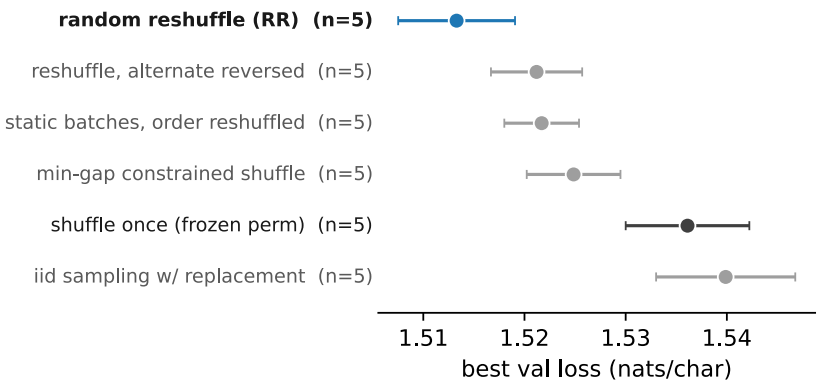

Figure 3.5: Best val loss per data-ordering regime in the toy testbed over 5 sees.

over a toy model on a character predicting task. As we see in Figure 3.5 no method ended up outperforming RR. In the assumption that the dev behavior might be caused by a few very closely following examples we devised the min-gap sampler, which would basically forbids a sample from repeating within n (n=). However as wee see in Figure 3.6, this did not inhibit the sawtooth. Following these experiments we suspected it might be due to average per batch

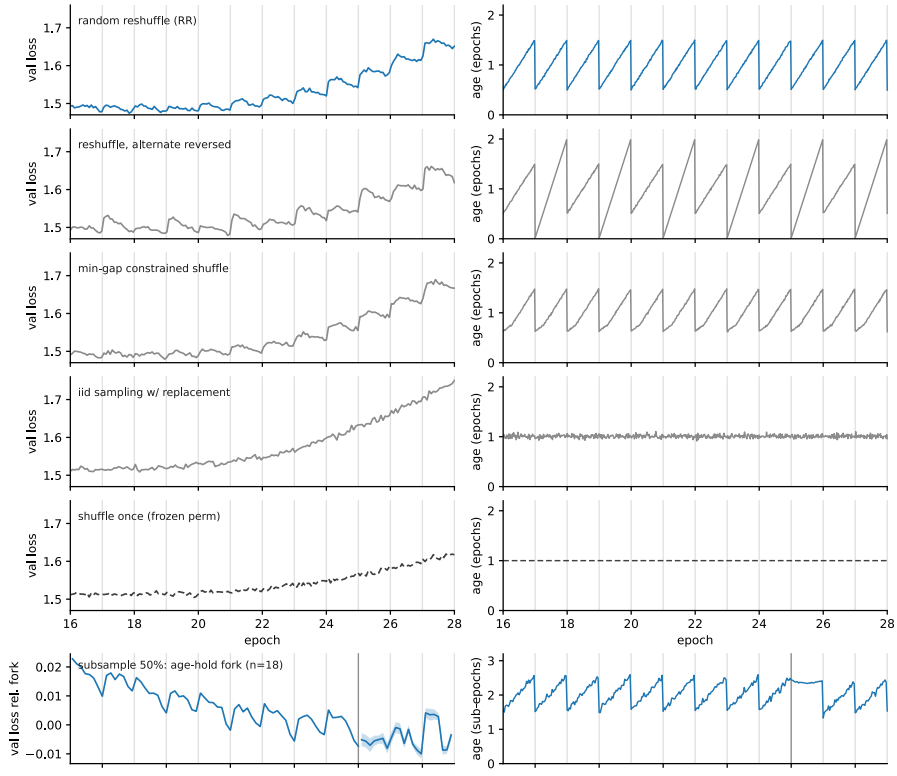

Figure 3.6: A contrast between the average age and dev loss developments.

recency. As seen in Figure 3.6 the average per batch recency gap corresponds perfectly to the trends we see in the different samplers. The RR gap goes up from 0.5 epochs to 1.5 epochs during the course of an epoch. To confirm our suspicions we ran an additional experiment where we controlled the average recency gap of an intermediate epoch to stay equal to high. This

experiment can be seen at the bottom of the figure, and shows that the sawtooth pattern indeed breaks in that instance.

How does age affect the models behavior?

After confirming that the average sample age is an important factor, we analyzed what layers seem to be most impacted. For this we looked at the effective update over a single epoch and found the layer norm to be especially impacted, as seen in Figure 3.7

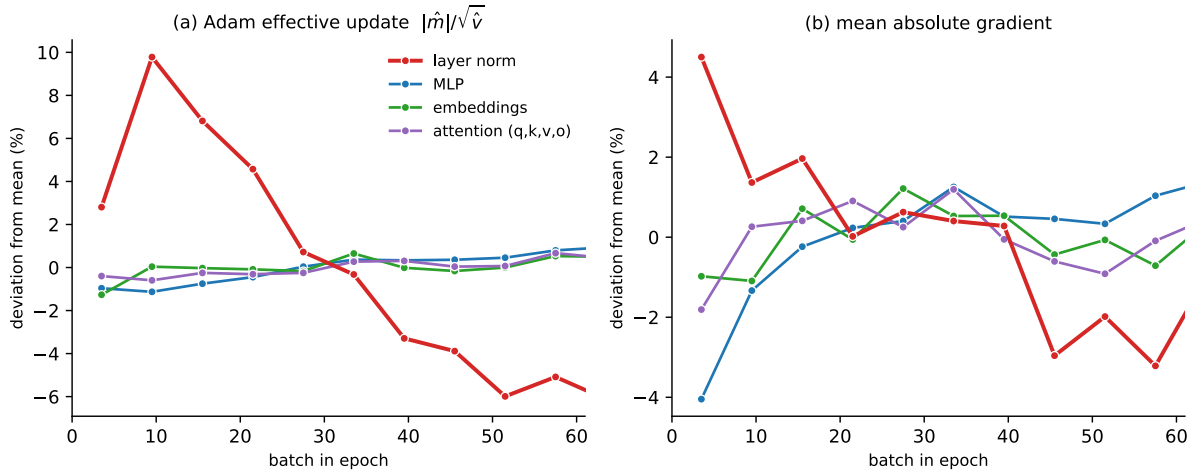

Figure 3.7: Adam effective update $|\hat{m}|/\sqrt{\hat{v}}$ (a) and mean absolute gradient (b) per parameter group, binned by position within the epoch. This was generated over the Toy Testbed (reshuffling, 3 seeds, epochs 10-30, normalized by its own mean).

3.5 Weight veraging as a Phase-Invariant Estimator

One downside of a single training run is that all models will have a bias towards some samples within the epoch depending on order. To mitigate this we proposed using an EMA over weights, such as a weight averaging method akin to SWA (Izmailov et al., 2018). We show that averaging these models in our experiments expands the law: $lr \times wd$ is stable for optima in a run, to $lr \times wd$ results in the same optimal loss independent of what wd or lr is chosen.

3.5.1 Methodology

Weight veraging EM A decaying learning rate scheduler preforms two jobs at once, on one hand it reduces the learning rate, effectively reducing the noise applied to the final model. At the same time it reduces the noise of the optimizer loss landscape, so the optimizer can fit to sharper minima. Weight averaging seperates these two effects and only applies a noise reduction to the final model, leaving the optimization noise constant. It can be argued that in an overfitting prone regime, a sharpening of the optimization landscape is not optimal.

Implementation Each run maintains a set of running averages of the model weights, updated after every optimizer step. These averages correspond to EMAs with different values $\in \{0\ 998, 0\ 999, 0\ 9995\}$.

$$v_0 = 0, \qquad v_t = \quad v_t \ _1 + \ 1 \qquad) w_t, \qquad w_t = v_t / \ 1 \qquad ^t)$$

The weights are read out with an Adam style bias correction, to avoid them being dragged down towards zero in the beginning.

Hyperparameter Search We use the original 12.5M hyperparameter data and the 48M and 286M models. At 48M parameters we run $lr \in \{1\ 75, 3\ 5, 7, 14, 28\} \times 10^{\ 4}$ and $wd \in \{0\ 417, 0\ 833, 1\ 667, 3\ 333\}$; at 286M $lr \in \{0\ 75, 1\ 5, 3, 6, 12\} \times 10^{\ 4}$ and$wd \in \{0\ 625, 1\ 25, 2\ 5, 5\}$. All other settings follow the main protocol (Sec. 3.3.1) and use seed 42. Unlike the cosine grids, each cell trains until it converges, evaluations are spaced every 2.5M/5M tokens for the 48m/286M models. Stopping occurs when the best averaged loss has not improved by $10^{\ 4}$ nats for six consecutive evaluations, with a cap of 40 epochs.

3.5.2 Results

EM models behavior depends largely on $lr \times wd$

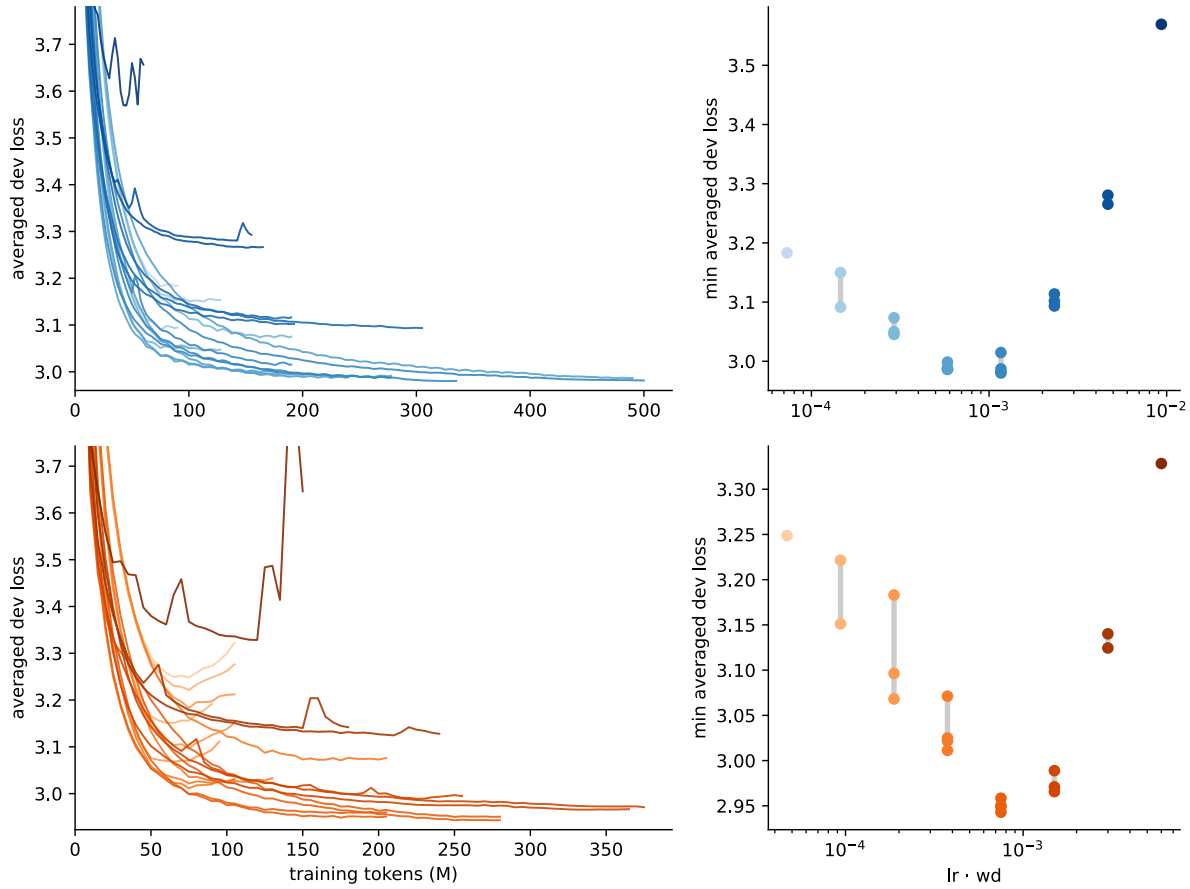

Figure 3.8: The averaged estimator collapses the grid onto the product law.

When training the EMA models we noticed that optimal loss value seemed to overwhelmingly depend on the $lr \times wd$ product, as seen in Figure 3.5.2. We can see that both the 52M and 286M model reduce to a law solely dependent on $lr \times wd$.

Internal behavior of EM models

To verify the dynamics at play we look at the per parameter group weight movement compared to overall weight magnitude. This approximately corresponds to how quickly a function changes. We see that the normalized groups converge to the same effective learning rate within a few steps. In Figure 3.9 we can see that in the top row attention is an outlier with consistently separate behavior. Since this stability is a result of matrix normalization (Kosson et al., 2024), we opted to apply DeepSeek Query-Key (QK) normalization (DeepSeek-AI, Liu, et al., 2025). As seen in the bottom row the attention started aligning as well. The QK normalization brought average gains, and reduced loss spread Table 3.11.

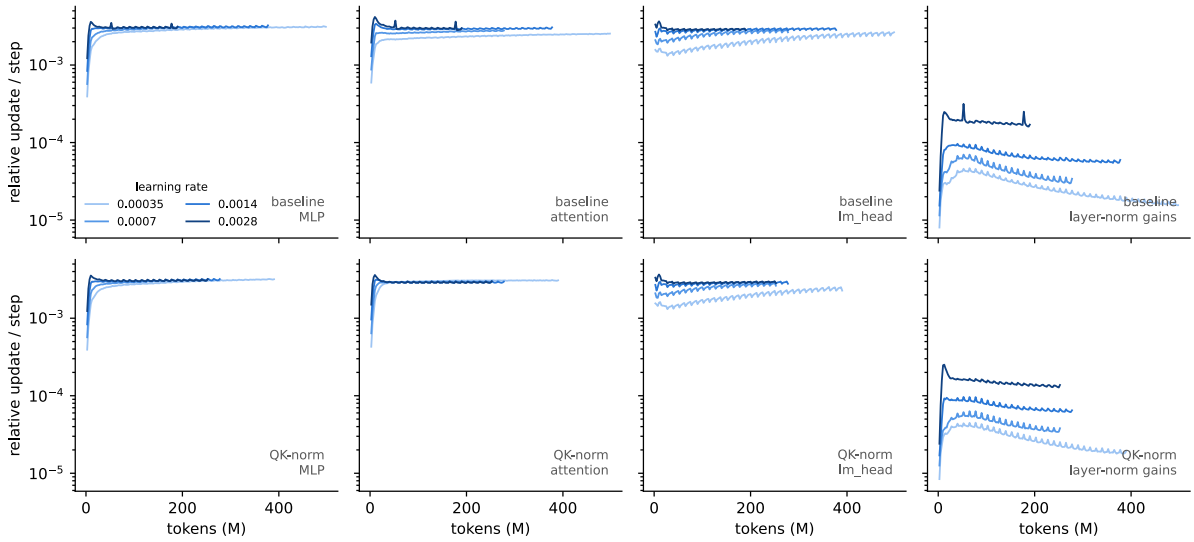

Figure 3.9: Stationary per-group effective learning rate along the iso-line $lr \cdot wd = 1\ 17 \times 10^{\ 3}$

	min averaged dev loss at η				
Architecture	3 5e 4	7e 4	1 4e 3	2 8e 3	spread
baseline	2.981	2.987	2.979	3.015	0.036
QK-norm	2.983	2.987	2.983	2.980	**0.008**

Table 3.11: Four test runs along the iso-product line $lr \times wd = 1\ 167 \times 10^{\ 3}$, with the learning rate varied $\times 8$ and the weight decay compensating (48M model). We see that the QK norm stabilised the trend further.

EM outperforms the standard Hyperparameter searched models

EMA runs outperformed hyperparameter tuned models by a significant margin, as seen in Table 3.12. We ran all performance-based zero-shot BabyLM methods, according to which the best EMA model trained would be ranked in 2nd place out of the 121 submissions made to the BabyLM 2026 challenge.

#	Model	BLiMP	Suppl.	EWoK	COMPS	EntTrack	Avg
1	*(Rank 1)* BabySteps_MurphysLaw-10M-mixed	71.60	63.93	51.94	53.13	27.95	**53.71**
2	255M champ, 12.5M const-LR (EMA)	73.99	64.34	53.68	53.56	22.82	53.68
3	**10M×10ep LTG, wd 3.62 (EMA)**	**73.84**	**66.03**	**53.50**	**53.74**	**19.37**	**53.29**
4	**10M×10ep LTG, wd 5 (EMA)**	**73.27**	**65.67**	**52.91**	**53.17**	**20.43**	**53.09**
5	Cosine 286M — 25M×4, lr 6.4e-4, wd 1.77	74.30	63.21	53.08	53.15	21.65	53.08
6	*(Rank 2)* Wordpiece-24-4	70.20	**66.52**	52.10	54.04	21.82	52.94
7	Cosine 99M — 100M×1, lr 8.3e-4, wd 0.5	74.98	62.07	54.12	54.02	19.28	52.90
8	Cosine 48M — 100M×1, lr 1.6e-3, wd 0.35	73.79	63.31	52.94	53.29	20.93	52.85
9	Cosine 286M — 100M×1, lr 4.8e-4, wd 0.77	75.37	61.89	52.73	**54.38**	19.72	52.82
10	*(Rank 3)* wwm_curriculum_simplification_40k	67.20	56.01	**56.07**	53.57	**28.45**	52.26
11	Cosine 286M — 12.5M×8, lr 1.2e-3, wd 2.5	71.09	61.48	53.24	52.31	20.53	51.73

Table 3.12: Zero-shot benchmark results. **Avg** is the unweighted mean of the five tasks. Best value per column in **bold**. The top three 2026 BabyLM models according to the metric tested were included in this table. The names of our runs that are BabyLM strict-small conform are **bolded**.

4 Loss ugmentation

4.1 Introduction

The best models in the BabyLM competitions (Warstadt, Mueller, et al., 2023) these past years have used some variation of masked language modeling (MLM) as a loss signal. This work proposes two main hypotheses for why MLM outperforms causal language modeling (CLM).

- MLM models have an advantage when it comes to solving minimal pair tasks, as they utilise information from the whole sentence when evaluating the correctness of a switched token.
- MLM generates more diverse permutations of the training samples. As an augmentation method, it allows for longer training without overfitting. Traditional MLM models, such as BERT (Devlin et al., 2019), were trained for 40 epochs, an unimaginable number for causal language models.

While MLM has performed extraordinarily well in past iterations of the BabyLM contest, the forward/backward pass restrictions introduced to the contest in 2025 impact MLM models disproportionately, as they (by convention) perform a forward pass over a complete sequence while 'only' using training signal from 15% of the tokens. We propose finding common ground between the MLM and CLM strategies by using the permutation language modeling (PLM) objective. In the PLM objective, the model predicts tokens over a permuted sequence.

- Like CLM, PLM theoretically allows every token to be predicted within a single forward/backward pass.
- Like MLM, PLM allows for the use of bidirectional information when predicting a single token, and significantly augments the training sequences.

In this work we will answer multiple questions:

- Traditional PLMs also train only on a subset of tokens, the last 16.66% to be precise: can we find a good permutation function which allows uninhibited training over every forward pass.
- Can we alter a simple Llama Model, to be used for PLM tasks?
- Can we achieve MLM like performance on the minimal pair tasks?

4.2 Related Works

XLNet XLNet (Z Yang et al., 2020) introduced the permutation model, where the sequence is traversed in random order. To do this the XLNet model needs to be aware of where the next target is without knowing what it is. To solve this the XLNet model introduced dial stream attention, where one residual stream knows where the next token is, but not the content. While XLNet doesn't directly use permuted sequences, it alters the mask such that the sequence is just transformer internally permuted.

GPT or BERT: why not both? GPT-BERT (L Charpentier and Samuel, 2024) gives direct evidence that in such small-data regimes the MLM and CLM objectives are complementary. The paper alters MLM to always predict the token one position to the right, so that the output at every position denotes the following token under either objective. This is what makes the causal datapoints efficient: under a standard MLM head a causal example would be solved by the identity, so supervision would have to be restricted to a masked minority of positions, whereas the shifted formulation supervises every position. Switching between the two modes then amounts to little more than masking the attention connections towards the future. Each datapoint is trained under exactly one of the two objectives, with the mixing ratio treated as a hyperparameter.

4.3 Methodology

PLM replaces causal left to right modeling with auto-regressive modeling over a permuted sequence. We try out two permutation functions, which are discussed in section 4.3.1. To model permuted sequences a Llama model is given positional knowledge, this is discussed in 4.3.2. When it comes to evaluating using the PLM models we try out Causal and Masked "modes", the specifics of which are explained in Section 2.3. The training settings and hyperparameters are detailed in Section 4.3.3.

4.3.1 Permuted sequences

We tried out two permutation functions, to provide the model with sequential information, which seems necessary for predictability of the asked for token, while at the same time allowing for more variability in the token sequences.

Gaussian jitter. Every position is perturbed independently.

$$k_i = i + \sigma\,\varepsilon_i, \qquad \varepsilon_i \sim \mathcal{N}\ 0,1), \tag{4.1}$$

The permutation is dense but local, the expected displacement grows with σ. Local context is perturbed but global context remains.

Token deferral. Each token is independently selected for deferral with probability p; a selected token is displaced *forward* by a distance d_i,

$$k_i = \begin{array}{ll} i, & \text{token kept,} \\ i + d_i, & \text{token deferred,} \end{array} \tag{4.2}$$

where a deferred token is placed at position $i + d_i$. Deferred tokens whose key exceeds the sequence length simply sort to the end. We study two distance distributions: a bounded uniform $d_i \sim \mathscr{U}\{1, \quad, 10\}$, and later an unbounded half-Gaussian $d_i = \max\ 1, \lceil |\mathscr{N}\ 0, \sigma_d)| \rfloor)$ with $\sigma_d = 20$, the latter designed so that retrieval distances at training time cover to hopefully be better at performing long range jumps. Throughout we use $p = 0\ 15$, matching the corruption rate of BERT-style masking.

geometry	keys	queries
SPLIT_ALL	original pos	all heads: next token pos
FUSED	original pos	9 heads: original pos, 6 heads: next token pos

Table 4.1: Per-head assignment of position tracks to queries and keys.

4.3.2 Shuffled positions in a Llama model

A standard decoder transformer has access to two separate streams of relevant information that a permutation model must separate.

- Original: its position in a sentence, using this information it can find information relevant to itself.
- Target: implicitly it knows the position of its target, it will be Original +1

Based on these indices we can form two different relevant distance functions:

- Current Token Context (key: original, query: original)
- Next Token Context (key: original, query: next token)

We try out two versions of this in the llama models as depicted in Table 4.1.

	architecture grid	final models
corpus (BabyLM 2025)	100 M-word track	strict-small (10 M words)
tokens	163 M	16.3 M
epochs	1	10
learning rate η	7×10^{-4}	3×10^{-4}
weight decay λ	0 1	2 5
schedule	cosine to zero, 600-step warmup	
batch	128 sequences $\times$ 128 tokens	
weight EMA	–	$= 0\ 999$
seeds	1	2

Table 4.2: Training configuration of the two model sets.

4.3.3 Training protocol

We train two sets of models. A first, exploratory set is trained for a single pass over the BabyLM 100 M-word track (163 M tokens under our 16k BPE tokenizer) to establish that learning under corruption is possible at all using the given permutation functions and Llama alterations. The final models are then trained for ten epochs on the BabyLM strict-small corpus. Training settings are described in Table 4.2.

4.3.4 Evaluation protocol

Causal-mode inference In causal mode the sequence passed into the model is not permuted, and the model scores the data like a normal next token prediction model.

Cloze-mode inference PLM and the deferral objective make cloze evaluation possible at no additional training. To score token x_i of a sentence bidirectionally, we present the sentence with x_i deferred to the final slot, and ask the model to predict it knowing all context. This helps depending on the minimal pair test, and is similar to the MLM pseudo log-likelihood (Salazar et al., 2020).

Per-paradigm mode selection. Because word-order paradigms favor causal scoring while agreement and licensing paradigms favor cloze scoring it makes sense to evaluate BLIMP using a mixture of methods. We evaluate BLIMP tasks using multiple PLM models and select a split based on majority vote.

	clean	jitter	deferral
SPLIT-ALL (all q on 1)	2.946	3.242	3.042
SPLIT-MIXED (9 on 0, 6 on 1)	–	3.268	3.068

Table 4.3: Comparison of query placement across evaluation conditions. SPLIT-ALL places all queries on next token distance, SPLIT-MIXED splits them 9/6 across self and next token distance.

4.4 Experiments

4.4.1 Which corruptions are trainable?

Single-pass comparison at 100 M words. Table 4.3 shows dev scores after the 100M runs have run over the two different model variants and the two permutation functions, as well as the clean control. Dev loss is evaluated using sequential data, so we can see than when it comes to learning causal language modeling jitter performs far worse.

4.4.2 What does the architecture need?

While the difference between jitter and deferral were clear, the difference between Split-All and Split-Mixed are much smaller. However Split-All slightly outperforms Split-Mixed as seen in Table 4.3. So for the rest of the paper we will be focusing on that modus.

4.4.3 Permutation as augmentation

Hypothesis 2 predicted that corrupted models might support more epochs before overfitting. According to our results, seen in Figure 4.1 the permuted models do train longer, and the deferred setting reaches comparable loss. Determining if hyper parameters might expose some latent benefit will be future work.

4.4.4 MLM-like inference from a causally trained model

Hypothesis 1 proposed that MLMs might have an advantagre in certain minimal pair tasks. Deferral training makes exactly this mode available at test time. Table 4.4 reports causal, cloze, fixed-assignment, and oracle scores on fast BLiMP. Firs, cloze does not dominate BLiMP completely, instead it seems different subtasks are more suited to the MLM and CLM perplexity metrics.

Tables 4.4 and 4.5 show that cloze outperforms on the BLIMP supplement task, but not BLIMP. Instead in BLIMP it seems that there are subsets that prefer one mode of evaluation. To create a static mapping and not overfit, we generate a "fixed" sorting of tasks into either cloze or mlm evaluation. In 4.4 we see that Fixed evaluation revocers almost all of the oracle benefits.

We calculated what tasks are the most biased toward one of the two evaluation methods in Table 4.6.

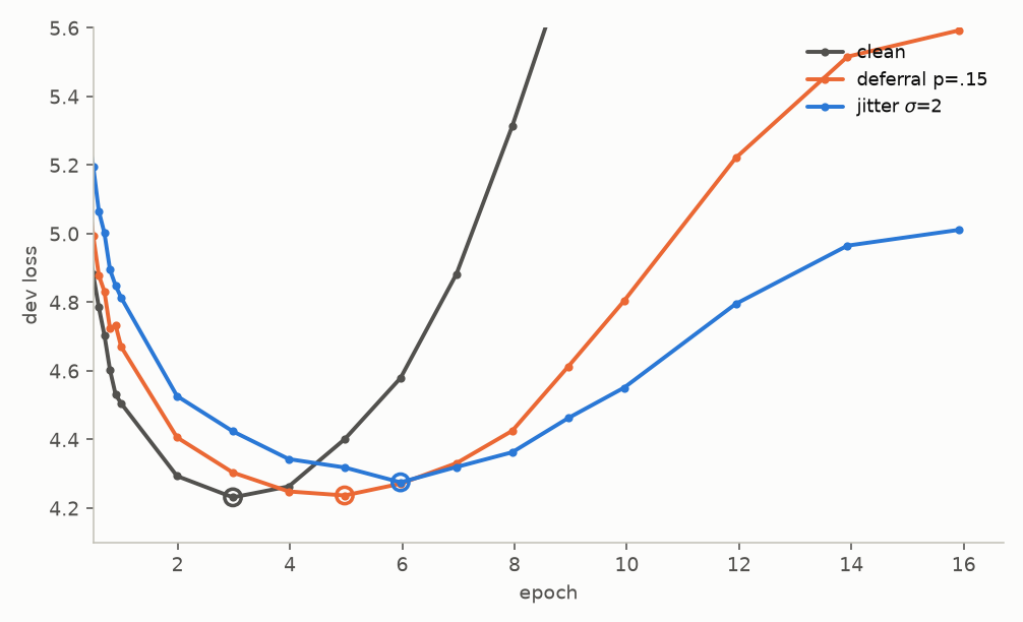

Figure 4.1: Held-out clean-text dev loss per epoch on the strict-small corpus (10 M words per epoch) for the three data schedules, trained with identical hyperparameters.

model	causal	cloze	fixed	oracle
deferral $\mathscr{U}\{1 \quad 10\}$ (10 M)	73.10	72.72	**76.27**	76.70
deferral half-Gaussian (10 M)	73.04	72.57	75.99	76.33

Table 4.4: Fast BLiMP under the two inference modes.

We did not detect any benefits in using cloze style evaluation for any EWOK or COMPS categories.

4.4.5 Final results

For the final BabyLM results we used the deferral dataset, testing both the uniform and gaussian deferral modes. All datasets are trained on the Split-All model. We provide both the raw causal scores as well as the "fixed" cloze/causal mixed scores for both BLIMP and BLIMP supplement. Results can be seen in Table 4.7. One of our models manages to outperform the currently top BabyLM 2026 model. It should be noted that our top model was evaluated using both cloze and causal evaluation metrics, fit to the BLIMP task. It is therefore possible that there is some test set leakage.

model	causal	cloze	oracle
deferral $\mathscr{U}\{1\ \ 10\}$	63.85	65.17	67.02
deferral half-Gaussian	64.59	**68.35**	**69.27**

Table 4.5: BLiMP supplement, cloze is strongly preferred.

paradigm	
largest causal advantage	
only_npi_licensor_present	+95 8
wh_island	+19 6
wh_questions_object_gap	+17 4
passive_2	+13 4
only_npi_scope	+13 2
largest cloze advantage	
matrix_question_npi_licensor_present	24 4
sentential_negation_npi_scope	19 0
wh_vs_that_with_gap_long_distance	18 1
existential_there_quantifiers_2	17 8
wh_vs_that_with_gap	13 1

Table 4.6: The five BLiMP paradigms with the largest advantage for each inference mode. is the mean causal cloze accuracy gap.

#	Model	BLiMP	Suppl.	EWoK	COMPS	EntTrack	Avg
1	**deferral Gaussian-20, s21 (EMA)**†	75.96	**69.27**	51.38	53.80	19.18	**53.92**
2	*(Rank 1)* BabySteps_MurphysLaw-10M-mixed	71.60	63.93	51.94	53.13	27.95	53.71
3	**deferral uniform-10, s42 (EMA)**†	**76.74**	65.19	51.32	53.84	18.61	53.14
4	*(Rank 2)* Wordpiece-24-4	70.20	66.52	52.10	54.04	21.82	52.94
5	**deferral uniform-10, s21 (EMA)**†	76.19	66.83	51.19	53.95	14.79	52.59
6	**deferral Gaussian-20, s42 (EMA)**†	76.26	66.18	51.48	53.37	14.92	52.44
7	**deferral Gaussian-20, s21 (EMA)**	72.90	64.59	51.38	53.80	19.18	52.37
8	**deferral uniform-10, s42 (EMA)**	73.68	63.97	51.32	53.84	18.61	52.28
9	*(Rank 3)* wwm_curriculum_simplification_40k	67.20	56.01	**56.07**	53.57	**28.45**	52.26
10	**deferral uniform-10, s21 (EMA)**	73.18	63.85	51.19	53.95	14.79	51.39
11	**deferral Gaussian-20, s42 (EMA)**	73.13	62.79	51.48	53.37	14.92	51.14

Table 4.7: Zero-shot benchmark results of the final deferral models against the top three 2026 BabyLM strict-small entries. Our models (all BabyLM strict-small conform) are **bolded**. Rows marked with † apply per-paradigm inference-mode selection (causal vs. cloze) to BLiMP and the supplement.

5 Conclusion

This thesis studied data-efficient pretraining, looking on one hand at hyperparameter settings advantageous for epoched settings, as well as data augmentation in the form of a PLM.

5.1 Summary of Findings

Weight decay in an integral part of epoched training, we confirmed that it interacts with learning rate to create a high effective learning rate necessary for regularization. We provided an example where it appeared that the Wang and Aitchison, 2025 $lr \times wd$ mapping stayed consistent as epochs were increased. We showed that weight averaging is a promising direction for epoched training. And offered an detailed look into the cyclic dev loss behavior. We offered a PLM permutation function that allows simultaneous training on CLM and MLM like objectives at the same time, and show this can be leveraged to achieve higher performance in certain evaluation settings, present in BLIMP tasks.

5.2 Limitations

The main sweeps use a single seed, and neighboring grid cells often differ by less than 0 01 nats. Optimal values should be rerun. We evaluated five zero-shot performance benchmarks, omitting GLUE and all human-likeness tasks, we do not claim this works models would achieve high overall scores. Finally, the per-paradigm cloze/causal mapping was derived from BLiMP itself, so part of the gain from 73 10 to 76 27 reflects selection on the test set.

Bibliography

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. 2023. GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints. arXiv: 2305.13245 [cs.CL].

BNC Consortium. 2007. The British National Corpus (BNC XML Edition). Oxford University Computing Services.

Lucas Charpentier and David Samuel. 2024. GPT or BERT: why not both? arXiv: 2410.24159 [cs.CL].

Lucas Georges Gabriel Charpentier and David Samuel. 2023. Not all layers are equally as important: Every Layer Counts BERT. ArXiv:2311.02265 [cs], November.

DeepSeek-AI, Daya Guo, et al. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. arXiv: 2501.12948 [cs.CL].

DeepSeek-AI, Aixin Liu, et al. 2025. DeepSeek-V3 Technical Report. arXiv: 2412.19437 [cs.CL].

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North merican Chapter of the ssociation for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers),* edited by Jill Burstein, Christy Doran, and Thamar Solorio, 4171–4186. Minneapolis, Minnesota: Association for Computational Linguistics, June.

Martin Gerlach and Francesc Font-Clos. 2018. A standardized Project Gutenberg corpus for statistical analysis of natural language and quantitative linguistics. *Computing Research Repository* arXiv:1812.08092.

Aaron Grattafiori et al. 2024. The Llama 3 Herd of Models. arXiv: 2407.21783 [cs. I].

Alexander Hägele, Elie Bakouch, Atli Kosson, Loubna Ben allal, Leandro Von Werra, and Martin Jaggi. 2024. Scaling Laws and Compute-Optimal Training Beyond Fixed Training Durations. In *The Thirty-eighth nnual Conference on Neural Information Processing Systems.*

Anna A. Ivanova, Aalok Sathe, Benjamin Lipkin, Unnathi Kumar, Setayesh Radkani, Thomas H. Clark, Carina Kauf, Jennifer Hu, R. T. Pramod, Gabriel Grand, Vivian Paulun, Maria Ryskina, Ekin Akyürek, Ethan Wilcox, Nafisa Rashid, Leshem Choshen, Roger Levy, Evelina Fedorenko, Joshua Tenenbaum, and Jacob Andreas. 2025. Elements of World Knowledge (EWoK): A Cognition-Inspired Framework for Evaluating Basic World Knowledge in Language Models. ArXiv:2405.09605 [cs], July.

Pavel Izmailov, Dmitrii Podoprikhin, T. Garipov, Dmitry P. Vetrov, and Andrew Gordon Wilson. 2018. Averaging Weights Leads to Wider Optima and Better Generalization. In *Conference on Uncertainty in rtificial Intelligence.*

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling Laws for Neural Language Models. arXiv: 2001.08361 [cs.LG].

Konwoo Kim, Suhas Kotha, Percy Liang, and Tatsunori Hashimoto. 2026. Pre-training under infinite compute. In *The Fourteenth International Conference on Learning Representations.*

Najoung Kim and Sebastian Schuster. 2023. Entity Tracking in Language Models. In *Proceedings of the 61st nnual Meeting of the ssociation for Computational Linguistics (Volume 1: Long Papers),* edited by Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, 3835–3855. Toronto, Canada: Association for Computational Linguistics, July.

Atli Kosson, Bettina Messmer, and Martin Jaggi. 2024. Rotational Equilibrium: How Weight Decay Balances Learning Across Neural Networks. arXiv: 2305.17212 [cs.LG].

Pierre Lison and Jörg Tiedemann. 2016. OpenSubtitles2016: Extracting Large Parallel Corpora from Movie and TV Subtitles. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16),* 923–929. Portorož, Slovenia: European Language Resources Association (ELRA), May.

Qi Liu and Wanjing Ma. 2025. The Epochal Sawtooth Phenomenon: Unveiling Training Loss Oscillations in Adam and Other Optimizers. arXiv: 2410.10056 [cs.LG].

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. arXiv: 1907.11692 [cs.CL].

Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations.*

Yucheng Lu, Wentao Guo, and Christopher De Sa. 2023. GraB: Finding Provably Better Data Permutations than Random Reshuffling. arXiv: 2205.10733 [cs.LG].

Brian MacWhinney. 2000. The CHILDES project: The database. Vol. 2. Psychology Press.

Kanishka Misra, Julia Rayz, and Allyson Ettinger. 2023. COMPS: Conceptual Minimal Pair Sentences for testing Robust Property Knowledge and its Inheritance in Pre-trained Language Models. In *Proceedings of the 17th Conference of the European Chapter of the ssociation for Computational Linguistics,* edited by Andreas Vlachos and Isabelle Augenstein, 2928–2949. Dubrovnik, Croatia: Association for Computational Linguistics, May.

Niklas Muennighoff, Alexander M. Rush, Boaz Barak, Teven Le Scao, Aleksandra Piktus, Nouamane Tazi, Sampo Pyysalo, Thomas Wolf, and Colin Raffel. 2025. Scaling Data-Constrained Language Models. arXiv: 2305.16264 [cs.CL].

Julian Salazar, Davis Liang, Toan Q. Nguyen, and Katrin Kirchhoff. 2020. Masked Language Model Scoring. In *Proceedings of the 58th nnual Meeting of the ssociation for Computational Linguistics,* edited by Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, 2699–2712. Online: Association for Computational Linguistics, July.

David Samuel, Andrey Kutuzov, Lilja Øvrelid, and Erik Velldal. 2023. Trained on 100 million words and still in shape: BERT meets British National Corpus. arXiv: 2303.09859 [cs.CL].

Andreas Stolcke, Klaus Ries, Noah Coccaro, Elizabeth Shriberg, Rebecca Bates, Daniel Jurafsky, Paul Taylor, Rachel Martin, Marie Meteer, and Carol Van Ess-Dykema. 2000. Dialogue Act Modeling for Automatic Tagging and Recognition of Conversational Speech. *Computational Linguistics* 26 (3): 339–371.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. 2024. RoFormer: Enhanced transformer with Rotary Position Embedding. *Neurocomput.* (NLD) 568, no. C (February).

Jean-Loup Tastet and Inar Timiryasov. 2024. BabyLlama-2: Ensemble-Distilled Models Consistently Outperform Teachers With Limited Data [in en]. ArXiv:2409.17312 [cs], September.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *dvances in Neural Information Processing Systems,* edited by I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, vol. 30. Curran Associates, Inc.

Xi Wang and Laurence Aitchison. 2025. How to set AdamW's weight decay as you scale model and dataset size. In *Forty-second International Conference on Machine Learning.*

Alex Warstadt, Aaron Mueller, Leshem Choshen, Ethan Wilcox, Chengxu Zhuang, Juan Ciro, Rafael Mosquera, Bhargavi Paranjabe, Adina Williams, Tal Linzen, and Ryan Cotterell. 2023. Findings of the BabyLM Challenge: Sample-Efficient Pretraining on Developmentally Plausible Corpora. In *Proceedings of the BabyLM Challenge at the 27th Conference on Computational Natural Language Learning,* 1–6. Association for Computational Linguistics.

Alex Warstadt, Alicia Parrish, Haokun Liu, Anhad Mohananey, Wei Peng, Sheng-Fu Wang, and Samuel R. Bowman. 2023. BLiMP: The Benchmark of Linguistic Minimal Pairs for English. ArXiv:1912.00582 [cs], February.

Greg Yang, Edward J. Hu, Igor Babuschkin, Szymon Sidor, David Farhi, Jakub Pachocki, Xiaodong Liu, Weizhu Chen, and Jianfeng Gao. 2022. Tensor Programs V: Tuning Large Neural Networks via Zero-Shot Hyperparameter Transfer. In *NeurIPS 2021.*

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2020. XLNet: Generalized Autoregressive Pretraining for Language Understanding. arXiv: 1906.08237 [cs.CL].

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. XLNet: Generalized Autoregressive Pretraining for Language Understanding. In *dvances in Neural Information Processing Systems,* edited by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, vol. 32. Curran Associates, Inc.

ppendices

.1 Epoch Scaling

.1.1 Detailed per run Metrics

We decided to log more detailed metrics, to get a better idea of the model internals. Due to size constraints we opted to log metrics for a small subset of layers known to display different dynamics. Within these layers we logged specific details on the matrix level. Details are listed in Table A.1.

Table A.1: Tracked Components

Category	**Components / Matrices**
Model Components	`embed`, `lm_head`, `layer_{first,middle,last}`
Attention Matrices	`q_proj`, `k_proj`, `v_proj`, `o_proj`
MLP Matrices	`gate_proj`, `up_proj`, `down_proj`

To later possibly analyse the symptoms and behaviours of overfitting and excessive weight decay, we logged the weight norms, the actually applied gradient update size, and this update split up into how much of this is weight decay and how much is the result of training momentum. We also added a metric looking at what the cosine similarity between the gradient and weight decay update portions are. Names, and details are listed in Table A.2.

Table A.2: Weight and Update Metrics

Metric	**Description**
`weight_norm`	$\|\theta\|_2$: L2 norm of weights
`update_total`	$\|\ \ \theta\|_2$: L2 norm of total weight update
`update_grad`	L2 norm of gradient-based update component
`update_wd`	L2 norm of weight decay update component
`update_ratio_total`	$\|\ \ \theta\|_2/\|\theta\|_2$
`update_ratio_grad`	Gradient update norm / weight norm
`update_ratio_wd`	Weight decay update norm / weight norm
`wd_over_grad`	Ratio of weight decay to gradient update norms
`cosine_wd_grad`	Cosine similarity between WD and gradient updates

To get a better feeling for why applied gradients might spike or differ, we also log the current size of the optimizer moment and second moment. Details are listed in Table A.3.

Research has found that the stability of activations might play a large role in hyperparameter transferability. For this reason we logged the impact current weight changes have on the outgoing

Table A.3: Adam Optimizer State Metrics

Metric	Description
`momentum`	$\|m\|_2$: L2 norm of first moment
`second_moment`	$\|v\|_2$: L2 norm of second moment
`m_over_v`	$\|m/\ v+10\ ^{10})\|_2$
`m_over_v_eps`	$\|m/\ \sqrt{v}+\varepsilon)\|_2$: actual Adam update direction

activation norm. This was more expensive than the other metrics, so it was only logged every 100 training steps.

Table A.4: Activation Metrics

Metric	Description
`activation`	Layer-averaged activation norm
`activation_effect_total`	Total activation change from weight update
`activation_effect_grad`	Activation change from gradient-only update
`activation_effect_wd`	Activation change attributable to weight decay

Due to interesting behavior at epoch boundaries, the training logs were generated for 5 steps before and after every epoch boundary.

Dataset	12.5M x 8		25M x 4		50M x 2		100M x 1	
	Tokens (M)	%	Tokens (M)	%	Tokens (M)	%	Tokens (M)	%
CHILDES	4.56	36.5	9.53	38.1	19.63	39.3	39.17	39.2
BNC (spoken)	1.27	10.1	2.11	8.4	3.28	6.6	6.09	6.1
Gutenberg	2.64	21.1	5.68	22.7	10.87	21.7	21.75	21.8
OpenSubtitles	2.38	19.0	4.39	17.6	9.25	18.5	18.97	19.0
Simple Wikipedia	1.52	12.2	2.98	11.9	6.34	12.7	12.75	12.8
Switchboard	0.14	1.1	0.30	1.2	0.62	1.2	1.27	1.3
Total	12.50	100.0	25.00	100.0	50.00	100.0	100.00	100.0

Table A.5: Distribution of datasets across the differently sized subsampled training sets for Experiment 1. Token counts are shown in millions (M) alongside within-split proportions.

Algorithm 1 Adaptive Grid Refinement Hyperparameter Search

Require: Initial grid runs $\mathscr{R}$, convergence threshold τ, max iterations T
Ensure: Optimal hyperparameters (learning rate η^*), weight decay λ^*))

function ADAPTIVESEARCH($\mathscr{R}, \tau, T$)
  for $t = 1$ **to** T **do**
    $\eta^*, \lambda^*) \leftarrow \arg\min_{r \in \mathscr{R}}$ dev $r)$

    // Compute radius per direction, generate neighbors
    for $d \in \{\eta^+, \eta$ $, \lambda^+, \lambda$ $\}$ **do**
      $\rho_d \leftarrow \min_{r \in \mathscr{R}} \| r$ $\mathbf{p}^* \|_d$
      $\mathbf{n}_d \leftarrow \mathbf{p}^* + \rho_d \cdot \mathbf{e}_d$
    end for

    // Check convergence
    if $\max_d \rho_d \leq \log$ $\tau)$ **then**
      return $\eta^*, \lambda^*)$
    end if

    // Train uncovered directions
    for $d \in \{\eta^+, \eta$ $, \lambda^+, \lambda$ $\}$ **do**
      if $\neg$COVERED $\mathbf{p}^*, \mathbf{n}_d, \mathscr{R})$ **then**
        $\mathscr{R} \leftarrow \mathscr{R} \cup \{$TRAIN $\mathbf{n}_d)\}$
      end if
    end for
  end for
  return failure
end function

function COVERED($\mathbf{p}^*, \mathbf{n}, \mathscr{R}$)
  return $\exists r \in \mathscr{R}$: r between $\mathbf{p}^*$ and $\mathbf{n}$ along axis
end function

Eidesstattliche Erkl rung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit im Masterstudiengang Informatik selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellen- verzeichnis nicht benannten Internet-Quellen – benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe. Sofern im Zuge der Erstellung der vorliegenden Abschlussarbeit generative Künstliche Intelligenz (gKI) basierte elektronische Hilfsmittel verwendet wurden, versichere ich, dass meine eigene Leistung im Vordergrund stand und dass eine vollständige Dokumentation aller verwendeten Hilfsmittel gemäß der Guten Wissenschaftlichen Praxis vorliegt. Ich trage die Verantwortung für eventuell durch die gKI generierte fehlerhafte oder verzerrte Inhalte, fehlerhafte Referenzen, Verstöße gegen das Datenschutz- und Urheberrecht oder Plagiate.

Unterschrift: *Ort, Datum:* Baltimore 17 August 2026